Databases

UNIT 4

Chapter 1

ER Model and Relational Model

LEARNING OBJECTIVES

- *Data model*
- *Schemas*
- *Three-schema architecture*
- *ER model*
- *Types of attributes*
- *Mapping cardinality*
- *Complex attributes*
- *Entity types, entity sets and value sets*
- *Weak entity set*
- *Relational database*
- *NULL in tuples*
- *Inherent constraint*
- *Referential and entity integrity constraint*

Introduction

A database is a collection of related data. By data, we mean facts that can be recorded and that have implicit meaning.

Example: Consider the names, telephone numbers and addresses of the people. We can record this data in an indexed address book and store it as Excel file on a hard drive using a personal computer. This is a collection of related data with an implicit meaning and hence is a database.

A database management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating and sharing databases among various users and applications.

1. Defining the database involves specifying the data types, structures, and constraints for the data to be stored in the database.
2. Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS
3. Manipulating a database includes such functions as querying the database to retrieve specific data, updating the database to reflect changes.
4. Sharing a database allows multiple users and programs to access the database concurrently.
5. Fundamental characteristics of the database approach is that it provides some level of data abstraction by hiding details of data storage that are not needed by users.

Data Model

A data model is a collection of concepts that can be used to describe the structure of a database. It provides the necessary means to achieve abstraction.

Schemas

In any data model, it is important to distinguish between the description of the database and the database itself. The description of a database is called the *database schema*, which is specified during database design and is not expected to change frequently.

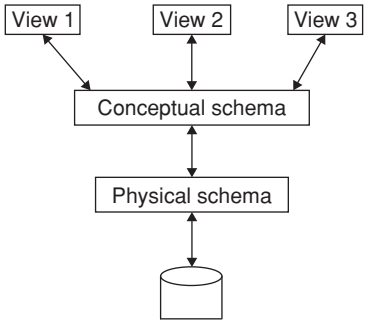
The actual data in a database may change frequently, for example the student database changes every time we add a student or enter a new grade for a student. The data in the database at a particular moment in time is called a *database state* or *snapshot*. It is also called the *current set of occurrences* or *instances in the database*.

The distinction between database schema and database state is very important. When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the empty state with no data. We get the initial state of the database when the database is first loaded with the initial data. The DBMS stores the description of the schema constructs and constraints, also called the *metadata* in the DBMS catalog so that DBMS software can refer to the schema whenever it needs. The schema is sometimes called the *intension*, and the database state an *extension* of the schema.

Three-schema Architecture

The goal of the three-schema architecture is to separate the user applications and the physical database.

Levels of Abstraction



1. The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database.
2. The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints.
3. The internal level has an internal schema, which describes the physical storage structures of the database. It describes the complete details of data storage and access paths for the database.

ER Model

Entity relationship model is a popular high-level conceptual data model. This model and its variations are used for the conceptual design of database applications, and many database design tools employ its concepts. ER model describes data as entities, relationships and attributes. The basic object that the ER model represents is an entity.

Entity

It is an object that exists and is distinguishable from other objects
(or)
Entity is a "thing" in the real world with an independent existence.
(or)
An entity is something that has a distinct, separate existence, although it need not be a material existence. In particular, abstractions and legal functions are usually regarded as entities. In general, there is no presumption that an entity is animate.

1. An object with a physical existence
   Example: A particular person, car, house, employee.
2. An object with a conceptual existence
   Example: A company, a job, a university course. Each entity has attributes, the particular properties that describe it.
   Example: An employee entity can be described by employee's name, age, address, salary and job.

Entity Set

Set of entities of same type that shares the same properties.

Example: All persons, all companies etc.

Example: Entity sets of customer and loan

Table 1 *Customer Entity Set*

Customer-id	Cust-name	Cust-street	City
C-143	John	MG Road	Sec.bad
C-174	Mary	SP Road	Hyd.bad
C-183	Tony	KD Road	Sec.bad
C-192	Satya	SG Road	Eluru

Table 2 *Loan Entity Set*

Loan-no	Amount
L-30	$3000
L-31	$4000
L-32	$3500
L-33	$4500
L-34	$5000

An entity is represented by a set of attributes and by a descriptive properties possessed by all members of an entity set.

Types of Attributes

1. Simple versus composite
2. Single valued versus multivalued
3. Stored versus derived.

Composite attribute: Composite attributes can be divided into smaller subparts, which represent more basic attribute that has their own meaning (Figure 1).

Example: A common example is Address, it can be broken down into a number of subparts such as street address, city, postal code; street address is further broken down into Number, street Name and Apartment number.

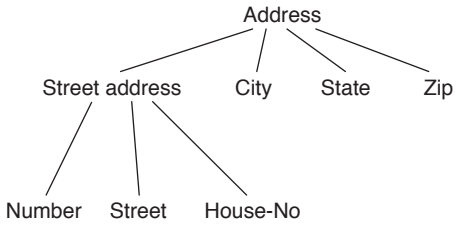


Figure 1 A hierarchy of composite attributes.

Street address is a composite attribute. Attributes that are not divisible are called simple (or) atomic attributes.

Single-valued versus multivalued attributes: Most attributes have a single value for a particular entity, such attributes are called *single-valued attribute*.

Example: Age is a single-valued attribute

Multivalued attributes: An attribute can have a set of values for the same entity.

Example: College degrees attribute for a person
Example: Name is also a multivalued attribute (Figure 2).

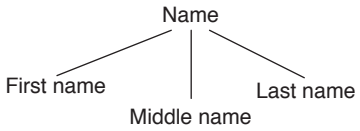


Figure 2 Multivalued attribute.

Stored versus derived attributes: Two (or) more attribute values are related.

Example: Age can be derived from a person's date of birth. The age attribute is called derived attribute and is said to be derivable from the DOB attribute, which is called a stored attribute.

Domain: The set of permitted values for each attribute.

Example: A person's age must be in the domain {0-130}

Relationship Sets

A relationship is an association among several entities. Relationship sets that involve two entity sets are binary. Generally, most relationships in databases are binary. Relationship sets may involve more than two entity sets.

Example: Employee of a bank may have responsibilities at multiple braches, with different jobs at different branches, then there is a ternary relation between employee, job and branch.

Mapping Cardinality

For a binary relationship set, mapping cardinality must be:

1. One-to-one
2. One-to-many
3. Many-to-one
4. Many-to-many

One-to-one: An entity in A is associated with at most one entity in B and an entity in B is associated with at most one entity in A (Figure 3).

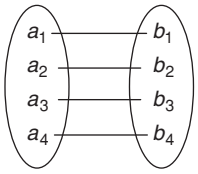


Figure 3 One-to-one relationship set.

One-to-many: An entity in A is associated with any number of entities in B. But an entity in B is associated with at most one entity in A (Figure 4).

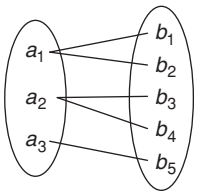


Figure 4 One-to-many relationship set.

Many-to-one: An entity in A is associated with at most one entity in B. But an entity in B can be associated with any number of entities in A (Figure 5).

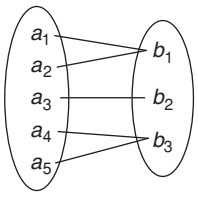


Figure 5 Many-to-one relationship set.

Many-to-many: An entity in A is associated with any number of entities in B. But an entity in B can be associated with any number of entities in A (Figure 6).

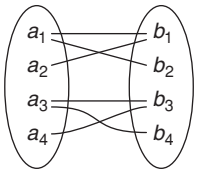
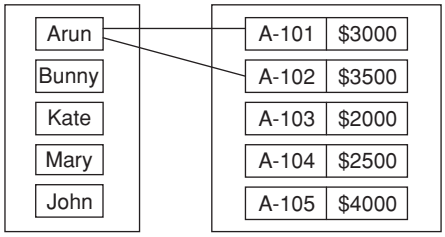


Figure 6 Many-to-many relationship set.

Example: One customer can have multiple accounts

Customer(c-Name) (Acc. no, Amount)

Table 3 *Example of One-to-many Relationship Set*



In Table 3, many-to-one relationship is not possible.

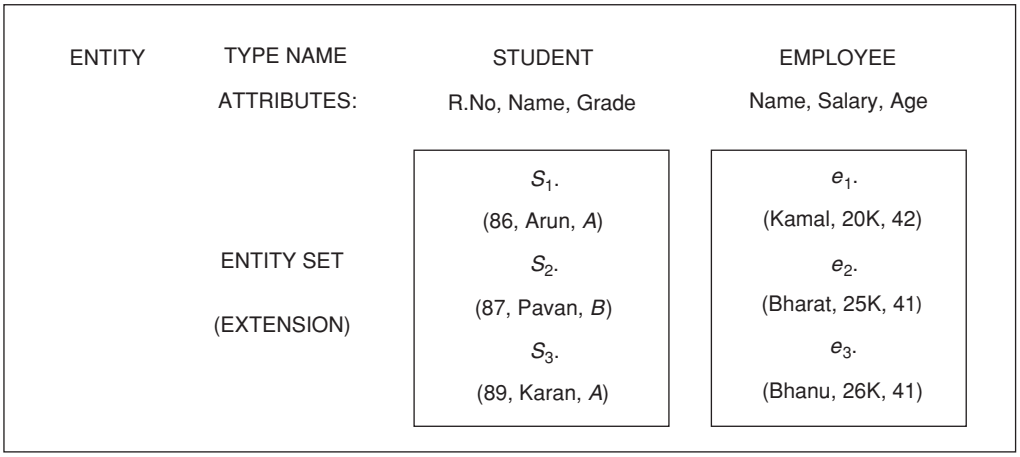
Complex Attributes

Composite and multivalued attributes can be nested in an arbitrary way. We can represent arbitrary nesting by grouping components of a composite attribute between parentheses () and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called complex attributes.

Example: A person can have more than one residence and each residence can have multiple phones, an attribute AddressPhone for a person can be specified as shown below. {AddressPhone ({Phone (Areacode, phoneNumber)}, Address (StreetAddress (StreetNumber, streetName, ApartmentNumber), city, state, zip))}

Entity types, entity sets and value sets

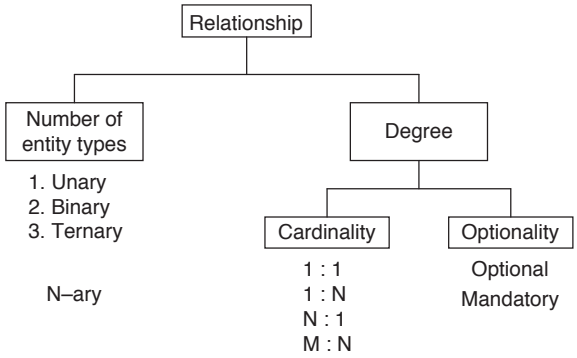
An entity type defines a collection of entities that have the same attributes. Each entity type in the database is described by its name and attribute. The following figure shows two entity types, named STUDENT and EMPLOYEE and a list of attributes for each



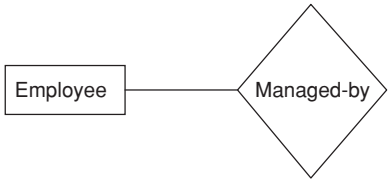
The collection of all entities of a particular entity type in the database at any point in time is called an *entity set*.

Types of relations

1. Unary relation.
2. Binary relation.
3. Ternary relation.
4. Quadnary relation.
5. N-ary relation



Unary relation If a relationship type is between entities in a single entity type then it is called a unary relationship type.

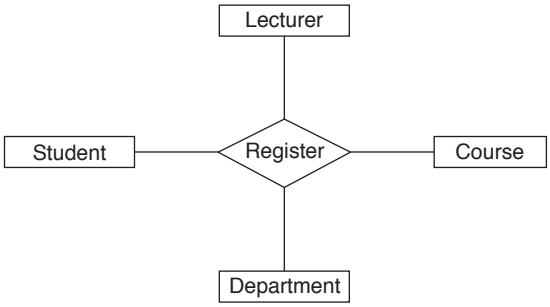


In employee entity, we will have all the employees including 'manager', this relation indicates, employees are managed by manager.

Binary relation If a relationship type is between entities in one type and entities in another type then it is called a *binary relation*, because two entity types are involved in the relation.

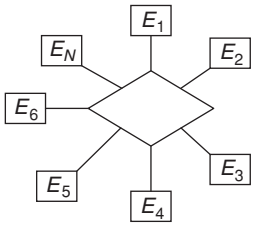
The above relation indicates that customers purchased product (or) products are purchased by customers.

Quadnary relation If a relationship type is among entities of four different types, then it is called *quadnary relation*.



In the above ER-diagram, any two entities can have a relation.

N-ary relation '*N*' number of entities will participate in a relation, and each entity can have a relation with all the other entities.

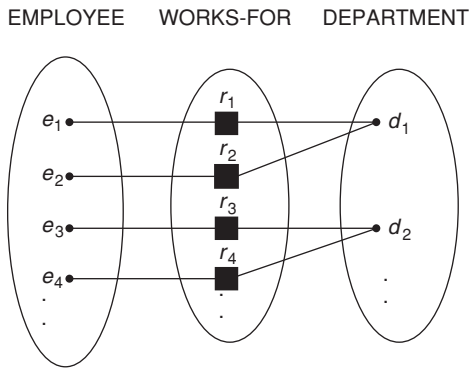


Cardinality Ratio and Participation Constraints

The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in

1. The participation constraint specifies whether the existence of an entity depends on its being related to another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in, and is some times called the *minimum cardinality constraint*.
2. There are two types of participation constraints:
   a. Total participation
   b. Partial participation

Example: If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates (or) works for at least one department.



Every entity in the EMPLOYEE set must be related to a DEPARTMENT entity via WORK-FOR. Total participation is also called *existence dependency*. If we do not expect every employee to manage a department, so the participation of EMPLOYEE in the relationship type is partial, means not necessarily all employees' entities are related to some department entity.

Cardinality ratio and participation constraints are taken together as the structural constraints of a relationship type. In ER diagrams, total participation is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line.

ER Diagrams (Figure 7 and 8)

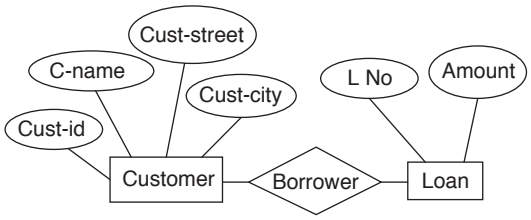


Figure 7 ER diagram.

Notations:

→ Represents entity

→ Represents relationship sets

→ Represents links between attributes to entity sets and entity sets to relationship sets

→ Represents attributes

→ Represents multivalued attributes

→ Represents primary key attribute

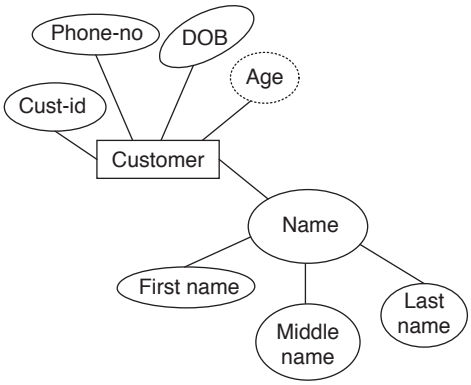
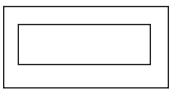


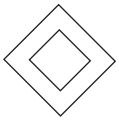
Figure 8 ER diagram.

→ Derived attribute

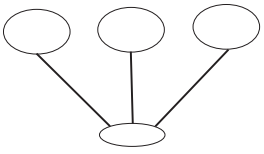
→ Multivalued attribute



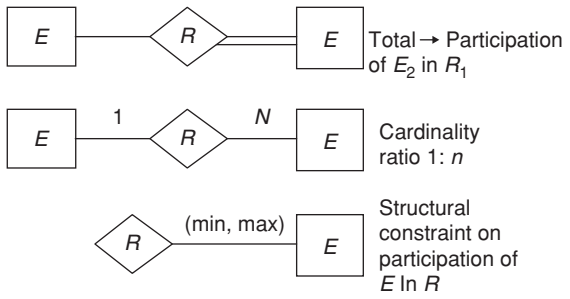
→ Weak entity



→ Weak entity set relation



→ Composite attribute



Cardinality Constraints

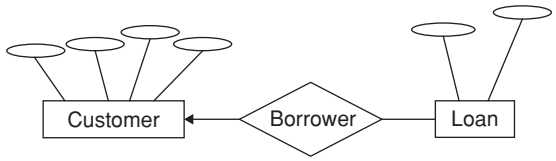
One-to-one

Each entity of one entity set is related to at most one entity of the other set. Only one matching record exists between two tables.

Example: Assume each owner is allowed to have only one dog and each dog must belong to one owner. The own relationship between dog and owner is one-to-one. One-to-one relationships can often combine the data into one table.

Examples:

1. One birdfeeder is located in one place in the yard.
2. One state has one governor.
3. One yard has one address.
4. One patient has one phone number.
2. One student has one ID.



One customer is associated to one loan via borrower

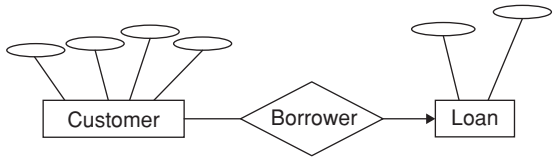
One-to-many

Examples:

1. One birdfeeder is visited by many birds.
2. One student can have many degrees.
3. One Book can be written by many authors.
4. One yard contains many bird feeders.
5. One patient has many prescriptions.

In the one-to-many relationship, a loan is associated with one customer via borrower.

Many-to-one

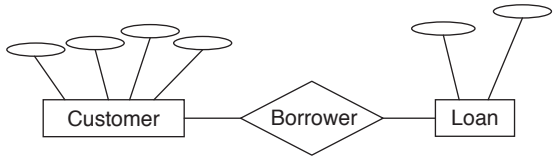


A customer is associated with at most one loan via borrower.

Many-to-many

Examples:

1. Many students are taught by many teachers.
2. Many patients are treated by many doctors.
3. Many medications are taken by many patients.
4. Many customers buy many products.
5. Many books are written by many authors.



Customer is associated with several loans and loan is associated with several customers.

ER Diagram with a Ternary Relationship

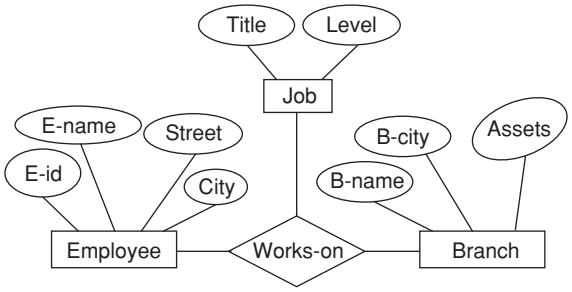


Figure 9 ER diagram with a ternary relationship.

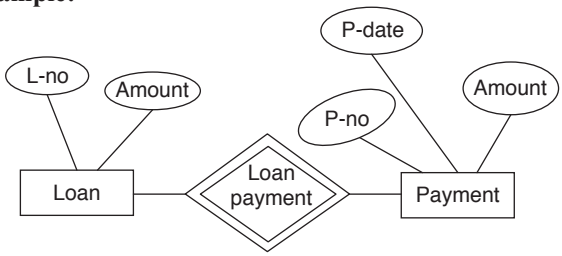
Weak Entity Set

An entity set that does not have a primary key is called *weak entity set*.

Weak entity set is represented by → 

Underline the primary key of a weak entity with a dashed line.

Example:



Relational Database

Relational Model

The relational model represents the database as a collection of relations. When a relation is thought of as a table of values, each row in the table represents a collection of related data values. Each row in the table represents a fact that typically corresponds to a real-world entity. The table name and

column names are used to help in interpreting the meaning of the values in each row.

In formal relational model terminology, a row is called a *tuple*, a column header is called an *attribute*, and the table is called a *relation*.

Domain

A domain is a set of atomic values. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain, to help in interpreting its values

Example:

1. *Set of telephone numbers*: The set of valid numbers in a particular country.
2. *Employee id numbers*: The set of valid employee numbers in a company.
3. *Names*: The set of character strings that represent names of persons.
4. *Grade-point average*: Possible values of computed grade point averages, each must be real (floating point) number between 0 and 4
5. *Research department names*: The set of research department names in a specialization, such as computer science, chemistry and applied mathematics.
6. *Research department codes*: The set of Research department codes, such as CS, CHE, AM.

The preceding is called *logical definitions of domains*. The data type for research department names is set of character strings that represent valid department names. A domain is thus given a name, data type, and format. Additional information for interpreting the values of a domain can also be given, for example a numeric domain such as person weights should have the units of measurements, such as kilograms or pounds.

1. The relational model is often described as having the following three aspects:
   - *Structural aspect*: The data in the database is perceived by the user as tables.
   - *Integrity aspect*: Those tables has to satisfy certain integrity constraints.
   - *Manipulative aspect*: The operators available to the user for manipulating those tables, for purposes of data retrieval, these operators derive tables form tables, the most important operators are 'SELECT', 'PROJECT' and JOIN.

Relation Schema

A relation schema 'R' denoted by $R(A_1, A_2, \ldots A_n)$ is made up of a relation name R and a list of attributes $A_1, A_2, \ldots A_n$. Each Attribute A_i is the name of role played by some domain D in the relation schema R. D is called the domain of A_i and is denoted by dom(A_i).

A relation schema is used to describe a relation and R is called the name of this relation. The degree of a relation is the number of attributes 'n' of its relation schema.

Example:

A relation schema of degree '7', which describes an employee is given below:

EMPLOYEE (Name, EId, HomePhone, Address, Office phone, Age, Salary)

Using the data type of each attribute, the definition is written as:

EMPLOYEE (Name: String, EId: INT, Homephone: INT, Address: String, OfficePhone: String, Age: Real, Salary: INT)

For this relation schema, EMPLOYEE is the name of the relation, which has '7' attributes.

2. A relation 'r' of the relation schema $R(A_1, A_2 \ldots A_n)$, also denoted by $r(R)$, is a set of n-tuples. $r = \{t_1, t_2 \ldots t_m\}$. Each n-tuple 't' is an ordered list of n values $t = <V_1, V_2 \ldots N_n>$, where each value V_i, $1 \le i \ldots n$, is an element of dom(A_i) or is a special null value.
3. The ith value in tuple t, which corresponds to the attribute A_i, is referred to as $t[A_i]$.
4. The following figure shows EMPLOYEE relation. Each type in a relation represents a particular employee entity. We display the relation as a table where each tuple is shown as a row and each attribute corresponds to a column header, indicating a role or interpretation of the values in that column. Null values represent attributes whose values are unknown or do not exist for some individual EMPLOYEE tuple.

Employee

Name	EId	Home Phone	Address	Office Phone	Age	Salary
Mahesh	30-01	870-223366	Warangal	NULL	35	40 k
Ramesh	30-02	040-226633	Hyderabad	NULL	36	40 k
Suresh	30-03	040-663322	Kolkata	040-331123	35	42 k
Dinesh	30-04	040-772299	Bangalore	040-321643	36	40 k

Fig: The attributes and tuples of a relation EMPLOYEE. The earlier definition of a relation can be restated as follows:
A relation $r(R)$ is a mathematical relation of degree 'n' on the domains dom(A_1), …, dom(A_n), which is a subset of the Cartesian product of the domains that define R:

$$r(R) \subset (\text{dom}(A_1) \times \text{dom}(A_2) \ldots \times \text{dom}(A_n))$$

Characteristics of Relations

There are certain characteristics that make a relation different from a file or a table.

Ordering of tuples in a relation: A relation is defined as a set of tuples. Tuples in a relation do not have any particular order. In a file, records are stored on disk in order. This ordering indicates first, second, i^{th}, and last records in the file. Similarly, when we display a relation as a table, the rows are displayed in a certain order.

Tuple ordering is not part of a relation definition, because a relation attempts to represent facts at a logical or abstract level. Many logical orders can be specified on a relation. Tuples in the EMPLOYEE relation could be logically ordered by values of Name or by EID or by Age or by some other attribute. When a relation is implemented as a file or displayed as a table, a particular ordering may be specified on the records of the file or the rows of the table.

NULL in Tuples

Each value in a tuple is an atomic value; that is, it is not divisible into components. Hence, composite and multivalued attributes are not allowed. This model is sometimes called the *flat relational model*. Multivalued attributes must be represented by separate relations, and composite attributes are represented only by their simple component attributes in the basic relational model.

NULLS are used to represent the values of attributes that may be unknown or may not apply to tuple. For example, some student tuples have null in their office phones because they do not have an office. In this case, the meaning behind NULL is not applicable. If a student has a NULL for home phone, it means either he/she does not have a home phone or he/she has one but we do not know it, in this case the meaning of NULL is 'Unknown'.

Relational Model Constraints

In a relational database, there will be many relations, and the tuples in those relations are related in various ways. There are many restrictions or constraints on the actual values in a database state.

Constraints on database can generally be divided into three main categories as follows:

1. Constraints that are inherent in the data model, we call them *inherent model-based constraints*.
2. Constraints that can be directly expressed in the schemes of the data model, by specifying them in the DDL (Data Definition Language). We call these *schema-based constraints*.
3. Constraints that cannot be directly expressed in the schemas of the data model, and they must be expressed and enforced by the application programs are *application-based constraints*.

Inherent Constraint

The constraint that a relation cannot have duplicate tuples is an *inherent constraint*. Another important category of constraints is data dependencies, which include 'functional dependencies' and 'multivalued dependencies'. They are used mainly for testing the 'goodness' of the design of a relational database and are utilized in a process called normalization.

Schema-based Constraints

1. Domain constraints
2. Key constraints
3. Constraints on nulls (Not null constraint)
4. Entity integrity constraints
5. Referential integrity constraints
6. Unique constraint
7. Check constraint

Domain constraints

Domain constraints specify that within each tuple, the value of each attribute 'X' must be an atomic value from the domain dom(X).

The data types associated with domains include standard numeric data types for integers

1. Short integer
2. Integer
3. Long integer
4. Real numbers
   - Float
   - Double-precision float
5. Characters
6. Booleans
7. Fixed-length strings
8. Variable-length strings
9. Date, time, time stamp
10. Money data types

Key constraints

A relation is a set of tuples. All elements of a set are distinct; hence, all tuples in a relation must also be distinct. This means no two tuples can have the same combination of values for all their attributes.

1. There are other subsets of attributes of a relation schema R with the property that no two tuples in any relation state 'r' of R should have the same combination of values of these attributes.

   Suppose that we denote one subset of attributes by 'SK', then for two distinct tuples t_1 and t_2 in a relation state 'r' of R, we have the following constraint:

   $$t_1[\text{SK}] = t_2[\text{SK}]$$

2. Any such set of attributes SK is called a *super key* of the relation schema *R*.

   SK specifies a uniqueness constraint that no two distinct tuples in any state *r* or *R* can have the same value for SK.
3. Every relation has at least one default super key, the set of all its attributes. *A* key, '*K*' of a relation schema *R* is a super key of *R* with the additional property that removing any attributes '*X*' from *K* leaves a set of attributes *K*' that is not a super key of *R* any more.

A key satisfies the following two constraints:

1. Two distinct tuples in any state of the relation cannot have identical values for all the attributes in the key.
2. A super key from which we cannot remove any attributes and still have the uniqueness constraints mentioned in above condition is known as a *minimal super key*.

   The first condition applies to both keys and super keys. The second condition is required only for keys.

Example: Consider the employee relation in Page no. 9. The attribute set {EId} is a key of employee because no two employee tuples can have the same value for EId.

Any set of attributes that include EId will form a super key.

1. {EId, Homephone, Name}
2. {EId, Age, Salary}
3. {Name, EId, Address}

However, the super key {EId, Name, Age} is not a key of EMPLOYEE, because removing Name or age or both from the set leaves us with a super key. Any super key formed from a single attributes is also a key. A key with multiple attributes must require all its attributes to have the uniqueness property.

A relation schema may have more than one key. In that case, each of the keys is called a *candidate key*.

Example: Employee relation has three candidate keys.
{Name, EId, Homephone}

One of the candidate keys is chosen as primary key of the relation.

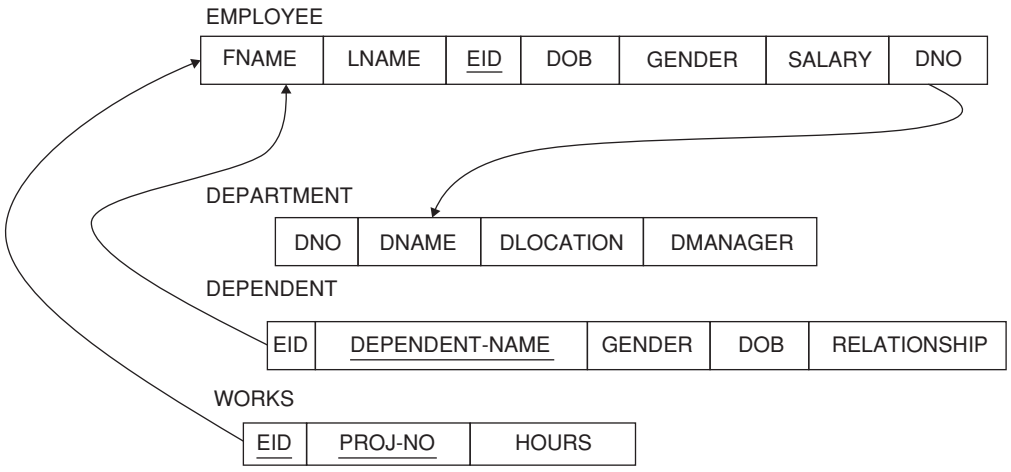
Another constraint on attributes specifies whether null values are permitted in tuples or not. If we want some tuples to have a valid (or) non-null value, we need to use NOT NULL constraint on that attribute.

Referential and entity integrity constraint

The entity Integrity constraint states that no primary key value can be null. If we have NULL values in the primary key column, we cannot identify some tuples in a relation.

1. Key constraints and entity Integrity constraints are specified on individual relations
2. Referential integrity constraint is specified between relations and used to maintain the consistency among tuples in the two relations.
3. Referential Integrity constraints states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation.
4. To understand the concept of Referential Integrity, first we have to understand the concept of FOREIGN KEY.
5. Suppose we have two relations R_1 and R_2. A set of attributes FK in relation schema R_1 is a foreign key of R_1 that references relation R_2 if it satisfies the following two rules:
   - The attributes in FK have the same domains as the primary key attributes PK of R_2, FK will have to refer to PK.
   - A value of FK in a tuple t_1 of the current state $r_1(R_1)$ either occurs as a value of PK for some tuple t_2 in the current state $r_2(R_2)$ or is null. We have t_1[FK] = t_2[PK], and we say that the tuple t_1 references to the tuple t_2. In this definition, R_1 is called the *referencing relation* and R_2 is the *referenced relation*.
6. A *foreign key* can refer to its own relation. We can diagrammatically display referential integrity constraints by drawing a directed Arc from each foreign key to the relation it references. The arrow head may point to the primary key of the referenced relation.

Example:



7. *Referential integrity rule*: The database must not contain any unmatched foreign key values.

If '*B*' References '*A*', then A must exist.

NOT NULL constraint

1. NOT NULL constraint restricts a column from having a NULL value. NOT NULL constraint can be applied to any column in a table.
2. We cannot give NULL values under that column
3. NOT NULL Constraint enforces a column to contain a proper value.
4. This constraint cannot be defined at table level.

Example: CREATE TABLE
student(RNo:INT Name:varchar(70)NOTNULL age:INT)
Suppose a row is inserted into the following table,

Insert into student values <11, NULL, 20>

In the schema, we enforced NOT NULL constraint on Name column, means Name cannot have NULL value, when the above insert command is executed, the system gives, NOT NULL constraint violation.

UNIQUE constraint

The column on which UNIQUE constraint is enforced should not have any duplicate values.

1. UNIQUE constraint can be enforced on any column except the primary key column.
2. By default primary key column will not accept any duplicate values that are handled by key constraint.
3. UNIQUE constraint can be applied at column level or table level.

Example: CREATE TABLE
student (RNo:INT Name:varchar(60) Grade:CHAR(1))
Assume that the table contains following tuples

Student

R no.	Name	Grade
11	Sita	B
12	Anu	A
13	Bala	A

Suppose the following tuple is inserted into the student table.

1. Insert into student values <14, 'Anu', '*B*'>
2. UNIQUE constraint is enforced on Name column, in the student table we have 'Anu', and again the new tuple contains name 'Anu', this Insert command violates the UNIQUE constraint.

CHECK constraint

This constraint is used to restrict a value of a column between a range.

1. When a value is inserted into particular column, before storing that, a check will be performed to see whether the values lie within the specified range.
2. If the value entered is out of range, it will not accept and violation happens.
3. It is like checking a condition before saving data into a column.

Example: Create table student (RNo:INT CHECK (Rno>0)
Name:varchar(60)
Dept:varchar(4))
Suppose the following tuple is inserted into student table.

1. Insert into student values <-4, 'Bhanu', 'CS'>
2. CHECK constraint is enforced on RNo column, the RNo should be greater than '0', but '-4' is given.
3. CHECK constraint is violated.

Creating table from a table: A view is called a *derived table* (or) *virtual table*, because the data stored in views is taken from already existing tables.

A view can also be defined as a logical subset of data from one or more tables.

Syntax:

```
CREATE    view view-name As
SELECT    column-names
FROM      table-name
WHERE     condition
```

Example: Consider the following table "sales".

Sales

Order-Id	Order Name	Previous Balance	Customer
21	Order 3	3000	Ana
22	Order 4	1000	Adam
23	Order 5	3000	Brat
24	Order 6	2000	John
25	Order 7	2000	Ana
26	Order 8	4000	Ana

Query to create a view:

```
CREATE view sales-view As
SELECT    *
FROM      Sales
WHERE     customer = 'Ana'
```

1. The data fetched from select statement will be stored in an object called 'sales-view'.
2. To display the contents stored in view, execute the following statement.

```
SELECT    *
FROM      Sales-view
```

Removal of specific rows:
Consider the following SQL query:

```
Delete *
FROM Sales
```

The above query will delete all the tuples from sales.

To remove specific rows, we have to specify the condition in WHERE clause.

Consider the table "sales" given in the above example.

Remove the rows from sales table whose previous balance is 3000.

SQL query:

```
Delete      *
FROM        Sales
WHERE       Previous-balance = 3000
```

Output:

Order Id	Order Name	Previous Balance	Customer
22	Order 4	1000	Adam
24	Order 6	2000	John
25	Order 7	2000	Ana
26	Order 8	4000	Ana

Referential actions: Referential Integrity can be violated if the value of any foreign key refers to a tuple that does not exist in the referenced relation.

When certain violations occur, we need to perform some alternate action. Those actions are as follows:

1. ON DELETE CASCADE
2. ON UPDATE CASCADE
3. ON DELETE SET NULL
4. ON DELETE SET DEFAULT

Example: Consider the given database:

SUPPLIERS:

Supplier Number	Supplier Name	Status	City
SN1	Suma	30	Hyderabad
SN2	Hari	20	Chennai
SN3	Anu	10	Hyderabad
SN4	Mahesh	20	Bombay
SN5	Kamal	30	Delhi

PARTS:

Part number	Part name	Colour	Weight	City
PN1	X	Red	13.0	Chennai
PN2	Y	Green	13.5	Bombay
PN3	X	Yellow	13.2	Hyderabad
PN4	Y	Green	14.1	Calcutta
PN5	Z	Red	14.3	Hyderabad
PN6	Z	Blue	14.2	Bombay

PROJECT:

Project Number	Project Name	City
PJ1	Display	Chennai
PJ2	OCR	Bombay
PJ3	RAID	Chennai
PJ4	SORTER	Hyderabad
PJ5	EDS	Chennai
PJ6	Tape	Bombay
PJ7	Console	Hyderabad

SHIPMENTS:

Supplier Number	Part Number	Project Number	Quantity
SN1	PN1	PJ1	300
SN1	PN1	PJ4	400
SN2	PN3	PJ1	350
SN2	PN3	PJ2	450
SN2	PN3	PJ3	640
SN2	PN3	PJ4	320
SN2	PN3	PJ5	330
SN2	PN3	PJ6	520
SN2	PN3	PJ7	480
SN2	PN5	PJ2	460
SN3	PN3	PJ1	440
SN3	PN4	PJ2	410
SN4	PN6	PJ3	310
SN4	PN6	PJ7	320
SN5	PN2	PJ2	340
SN5	PN2	PJ4	350
SN5	PN5	PJ5	360
SN5	PN5	PJ7	370
SN5	PN6	PJ2	380
SN5	PN1	PJ4	420
SN5	PN3	PJ4	440
SN5	PN4	PJ4	450
SN5	PN5	PJ4	400
SN5	PN6	PJ4	410

Consider the following statement:

```
DELETE FROM SUPPLIER
WHERE SUPPLIER – NUMBER = 'SN1'
```

It deletes the supplier tuple for supplier 'SN1'. The database has some other tables which have 'SN1' tuple (Shipments table). The application does not delete those suppliers, then it will find a violation, and an exception will be raised.

An alternate approach is possible, one that might be preferable in some cases, and that is for the system to perform an appropriate 'compensating action' that will guarantee that the overall result does still satisfy the constraint. In the example, the compensating action would be for the system to delete the shipments for supplier SN1 "automatically".

We can achieve this effect by extending the foreign key as indicated below:

```
CREATE TABLE SHIPMENT{.....}.....
FOREIGN KEY {SUPPLIER – NUMBER}
REFERENCES
SUPPLIER ON DELETE CASCADE
```

The specification ON DELETE CASCADE defines a delete rule for this particular foreign key, and the specification CASCADE is the referential action for that delete rule. The meaning of these specifications is that a DELETE operation on the suppliers relvar will 'Cascade" to delete matching tuples (if any) in the shipments relvar as well.

Same procedure is applied for all the referential actions.

Triggers

Triggers are precompiled procedures that are stored along with the database and invoked automatically whenever some specified event occurs.

Suppose we have a view called HYDERABAD - SUPPLIER defined as follows:

```
CREATE VIEW HYDERABAD-SUPPLIER
AS SELECT SUPPLIER - NUMBER, SUPPLIER-
NAME, STATUS
FROM SUPPLIER
WHERE CITY = 'HYDERABAD',
```

Normally, if the user tries to insert a row into this view, SQL will actually insert a row into the underlying base table SUPPLIERS with CITY value whatever the default is for the CITY column. Assuming that default is not Hyderabad, the net effect is that the new row will not appear in the view; therefore, let us create a triggered procedure as follows:

```
CREATE TRIGGER HYDERABAD -
SUPPLIER - INSERT
INSTEAD OF INSERT ON HYDERABAD
- SUPPLIER
REFERENCING NEW ROW AS R
FOR EACH ROW
INSERT INTO SUPPLIERS (SUPPLIER -
NUMBER, SUPPLIER - NAME, STATUS, CITY)
VALUES (R. SUPPLIER - NUMBER, R.
SUPPLIER - NAME, R. STATUS, 'HYDERABAD');
```

Inserting a row into the view will now cause a row to be inserted into the underlying base table with CITY value equal to Hyderabad inserted of the default value.

In general, CREATE TRIGGER specifies, among other things, an event, a condition, and an action.

The event is an operation on the database ("INSERT ON HYDERABAD - SUPPLIER" in the example)

1. The "condition" is a Boolean expression that has to evaluate to TRUE in order for the action to be executed.
2. The 'action' is the triggered procedure ("INSERT INTO SUPPLIERS …)
3. The event and condition together are sometimes called the *triggering event*. The combination of all three (event, condition, and action) is usually called a trigger.
4. Possible events include INSERT, DELETE, UPDATE, reaching end-of-transaction (COMMIT) reaching a specified time of day, exceeding a specified elapsed time, violating a specified constraint, etc.
5. A database that has associated triggers is sometimes called an active database.

Base Table Constraints

SQL-base table constraints are specified on either CERATE TABLE or ALTER TABLE. Each such constraint is a candidate key constraint, a foreign key constraint, or a CHECK constraint.

Candidate keys: An SQL candidate key definition takes one of the following two forms:

```
PRIMARY KEY (< column name comma list>)
UNIQUE (< column name comma list>)
The following example illustrates base table constraints of all three kinds:
CREATE TABLE SHIPMENTS
(SUPPLIER-NUMBER. SUPPLIER-NUMBER
NOT NULL, PART - NUMBER PART-NUMBER
NOT NULL, QUANTITY NOT NULL
PRIMARY KEY (SUPPLIER - NUMBER,
PART NUMBER)
FOREIGN KEY (SUPPLIER-NUMBER)
REFERENCES SUPPLIERS
ON DELETE CASCADE
ON UPDATE CASCADE,
FOREIGN KEY (PART-NUMBER)
REFERENCES PARTS
ON DELETE CASCADE
ON UPDATE CASCADE
CHECK(QUANTITY ≤ QUANTITY (0) AND
QUANTITY ≤ QUANTITY (1000));
```

A check constraint of the form CHECK (< column name > IS NOT NULL) can be replaced by a simple NOT NULL specification in the definition of the column.

EXERCISES

Practice Problems I

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Consider the following two tables T_1 and T_2. Show the output for the following operations:

Table T_1

P	Q	R
11	a	6
16	b	9
26	a	7

Table T_2

A	B	C
11	b	7
26	c	4
11	b	6

What is the number of tuples present in the result of given algebraic expressions?

(i) $T_1 \bowtie_{T1.P = T2.A} T_2$

(A) 2 (B) 3
(C) 4 (D) 5

(ii) $T_1 \bowtie_{T1.Q = T2.B} T2$

(A) 2 (B) 3
(C) 4 (D) .5

(iii) $T_1 \bowtie_{(T1.p = T2.A \text{ AND } T1.R = T2.C)} T_2$

(A) 1 (B) 2
(C) 3 (D) .4

2. Suppose $R_1(A, B)$ and $R_2(C, D)$ are two relation schemas. Let R_1 and R_2 be the corresponding relation instances. B is a foreign key that refers to C in R_2. If data in R_1 and R_2 satisfy referential integrity constraints, which of the following is true?

(A) $\pi_B(R_1) - \pi_c(R_2) = \phi$

(B) $\pi_C(R_2) - \pi_B(R_1) = \phi$

(C) $\pi_B(R_1) - \pi_C(R_2) \neq \phi$

(D) Both A and B

3. Consider the following relations:

A, B and C

A

Id	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

B

Id	Name	Age
15	Shreya	24
25	Hari	40
98	Rohit	20
99	Rohit	11

C

Id	Phone	Area
10	2200	02
99	2100	01

How many tuples does the result of the following relational algebra expression contain? Assume that the scheme of ($A \cup B$) is the same as that of A.

$(A \cup B) \bowtie_{A.Id>40 \ v \ c.Id<15} C$

(A) 6 (B) 7
(C) 8 (D) 9

4. Consider the relations A, B and C given in Question 3. How many tuples does the result of the following SQL query contain?

```
SELECT    A.Id
FROM      A
WHERE     A.Age>
ALL       (SELECT B.Age
FROM      B
WHERE     B.Name = 'Arun')?
```

(A) 0 (B) 1
(C) 2 (D) 3

5. Consider a database table T containing two columns X and Y each of type integer. After the creation of the table, one record ($X = 1$, $Y = 1$) is inserted in the table. Let MX and MY denote the respective maximum value of X and Y among all records in the table at any point in time. Using MX and MY, new records are inserted in the table 128 times with X and Y values being MX + 1, 2 * MY + 1, respectively. It may be noted that each time after the insertion, values of MX and MY change. What will be the output of the following SQL query after the steps mentioned above are carried out?

SELECT Y FROM T WHERE $X = 7$;?

(A) 15 (B) 31
(C) 63 (D) 127

6. Database table by name loan records is given below:

Borrower	Bank Manager	Loan Amount
Ramesh	Sunderajan	10000
Suresh	Ramgopal	5000
Mahesh	Sunderajan	7000

What is the output of the following SQL Query SELECT count (*)

FROM((SELECT Borrower, Bank-manager

FROM Loan-Records)AS S

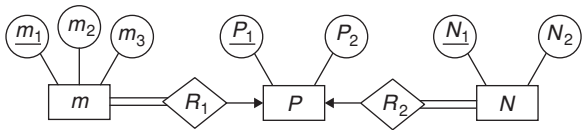
NATURAL JOIN

(SELECT Bank-manager, Loan-Amount

FROM Loan-Records) AS T);

(A) 3 (B) 4
(C) 5 (D) 6

7. Consider the following ER diagram:



What is the minimum number of tables needed to represent M, N, P, R_1, R_2?

(A) 2 (B) 3
(C) 4 (D) 5

8. Let E_1 and E_2 be two entities in an ER diagram with simple single-valued attributes. R_1 and R_2 are two relationships between E_1 and E_2, where R_1 is one-to-many and R_2 is many-to-many. R_1 and R_2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model?

(A) 2 (B) 3
(C) 4 (D) 5

9. The following table has two attributes A and C where A is the primary key and C is the foreign key referencing A with ON DELETE CASCADE.

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

What is the set of all tuples that must be additionally deleted to preserve referential integrity when the tuple (2, 4) is deleted?

(A) (5, 2),(7, 2) (B) (5, 2), (7, 2),(9, 5)
(C) (5, 2),(9, 5) (D) (2, 4), (7, 2)

10. Consider the following SQL query

SELECT DISTINCT $a_1, a_2, a_3 \ldots a_n$

FROM $R_1, R_2 \ldots R_m$

WHERE P

For any arbitrary predicate P, this query is equivalent to which relational algebra expression?

(A) $\Pi_{a_1, a_2, \ldots a_n}(\sigma_p R_1 \times R_2 \times \ldots \times R_m)$

(B) $\sigma_{a_1, a_2, \ldots a_n}(\sigma_p R_1 \times R_2 \times \ldots \times R_m)$

(C) $\sigma_{a_1, a_2, \ldots a_n}(\prod_p R_1 \times R_2 \times \ldots \times R_m)$

(D) $\Pi_{R_1, R_2, \ldots Rm}(\sigma_p a_1 \times a_2 \times \ldots \times (a_n)$

11. Consider the following relation schema pertaining to a student's database:

Student (Rollno, name, address)

Enroll (Rollno, courseno, coursename)

Where the primary keys are shown underlined. The number of tuples in the student and Enroll tables are 120 and 6, respectively. What are the maximum and minimum number of tuples that can be present in

(student * Enroll)

Where '*' denotes natural join?

(A) 6, 6 (B) 6, 120
(C) 120, 6 (D) 120, 120

12. A relational schema for a train reservation database is given below

Table 4 *Passenger*

Pid	P Name	Age
0	'Sachin'	65
1	'Rahul'	66
	'Sourav'	67
3	'Anil'	69

Table 5 *Reservation*

Pid	Class	Tid
0	AC	8200
1	AC	8201
2	SC	8201
5	AC	8203
1	SC	8204
3	AC	8202

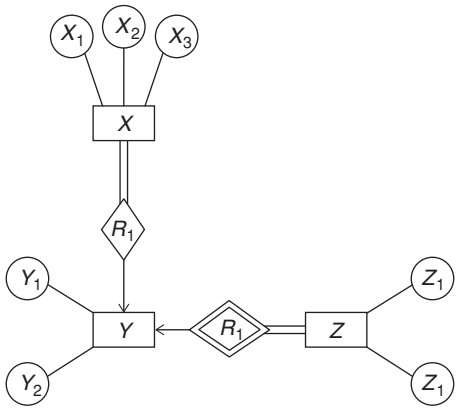
What pid's are returned by the following SQL query for the above instance of the tables?

```
SELECT    pid
FROM      Reservation
WHERE     class = 'AC' AND
          EXISTS (SELECT *
                  FROM passenger
          WHERE age > 65 AND
          Passenger.pid = Reservation.pid)
```

(A) 0, 1 (B) 1, 3
(C) 1, 5 (D) 0, 3

13. Given {customer} is a candidate key, [customer name, customer street} is another candidate key then
(A) {customer id, customer name} is also a candidate key.
(B) {customer id, customer street} is also a candidate key.
(C) {customer id, customer name, customer street} is also a candidate key.
(D) None

Common data for questions 14 and 15: Consider the following diagram,



14. The minimum number of tables needed to represent X, Y, Z, R_1, R_2 is
(A) 2 (B) 3
(C) 4 (D) 5

15. Which of the following is a correct attribute set for one of the tables for the correct answer to the above questions?
(A) $\{X_1, X_2, X_3, Y_1\}$ (B) $\{X_1, Y_1, Z_1, Z_2\}$
(C) $\{X_1, Y_1, Z_1\}$ (D) $\{M_1, Y_1\}$

16. UPDATE account SET
DA = basic * .2,
GROSS = basic * 1.3,
Where basic > 2000;
(A) The above query displays DA and gross for all those employees whose basic is ≥ 2000
(B) The above query displays DA and gross for all employees whose basic is less than 2000
(C) The above query displays DA as well as gross for all those employees whose basic is >2000
(D) Above all

17. Which of the following query transformations is correct?
R_1 and R_2 are relations C_1, C_2 are selection conditions and A_1 and A_2 are attributes of R_1
(A) $\sigma_{C1}(\sigma_{C1}(R_1)) \rightarrow \sigma_{C2}(\sigma_{C2}(R_1))$
(B) $\sigma_{C1}(\sigma_{A1}(R_1)) \rightarrow \sigma_{A1}(\sigma_{C1}(R_1))$
(C) $\pi_{A2}(\pi_{A}(R_1)) \rightarrow \pi_{A}(\pi_{A2}(R_1))$
(D) All the above

18. Consider the following query select distinct $a_1, a_2, \ldots a_n$ from $r_1, r_2 \ldots r_m$ where P for an arbitrary predicate P, this query is equivalent to which of the following relational algebra expressions:
(A) $\underset{a_1 \ldots a_n}{\pi}\ \sigma_p(r_1 \times r_2 \times \ldots \times r_m)$
(B) $\underset{a_1 \ldots a_n}{\pi}\ \sigma_p(r_1 \times r_2 \times r_3 \times \ldots . \times r_m)$
(C) $\underset{a_1 \ldots a_n}{\sigma}\ \pi_p(r_1 \times r_2 \times \ldots \times r_m)$
(D) $\underset{a_1 \ldots a_n}{\sigma}\ \pi_p(r_1 \times r_2 \times \ldots \times r_m)$

19. The relational algebra expression equivalent to the following tuple calculus expression
$\{a\}\ a \in r \wedge (a[A] = 10 \wedge a[B] = 20)$ is
(A) $\sigma_{(A = 10\ r\ B = 20)}$r
(B) $\sigma_{(A = 10)}(r) \cup \sigma_{(B = 20)}(r)$
(C) $\sigma_{(A = 10)}(r) \cap \sigma_{(B = 20)}(r)$
(D) $\sigma_{(A = 10)}(r) - \sigma_{(B = 20)}(r)$

20. Which of the following is/are wrong?
(A) An SQL query automatically eliminates duplicates.
(B) An SQL query will not work if there are no indexes on the relations.
(C) SQL permits attribute names to be repeated in the same relation
(D) All the above

Practice Problems 2

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. If $ABCDE$ are the attributes of a table and $ABCD$ is a super key and ABC is also super key then
(A) $A\ B\ C$ must be candidate key
(B) $A\ B\ C$ cannot be super key
(C) $A\ B\ C$ cannot be candidate key
(D) $A\ B\ C$ may be candidate key

2. The example of derived attribute is
(A) Name if age is given as other attribute
(B) Age if date_of_birth is given as other attribute
(C) Both (A) and (B)
(D) None

3. The weak entity set is represented by
(A) box
(B) ellipse
(C) diamond
(D) double outlined box

4. In entity relationship diagram double lines indicate
(A) Cardinality
(B) Relationship
(C) Partial participation
(D) Total participation

5. An edge between an entity set and a binary relationship set can have an associated minimum and maximum cardinality, shown in the form 1… *h* where 1 is the minimum and *h* is the maximum cardinality A minimum value 1 indicates:
(A) total participation (B) partial participation
(C) double participation (D) no participation

6. Let *R* be a relation schema. If we say that a subset *k* of *R* is a super key for *R*, we are restricting *R*, we are restricting consideration to relations *r*(*R*)in which no two district tuples have the same value on all attributes in *K*. That is if t_1 and t_2 are in *r* and $t_1 \uparrow t_2$
(A) $t_1[k] = 2t_2[K]$ (B) $t_2[K] = 2t_1[k]$
(C) $t_1[k] = t_2[k]$ (D) $t_1[k] \uparrow t_2[k]$

7. Which one is correct?
(A) Primary key ⊂ Super key ⊂ Candidate key
(B) Candidate key ⊂ Super key ⊂ Primary key
(C) Primary key ⊂ Candidate key ⊂ Super key
(D) Super key ⊂ Primary key ⊂ Candidate key

8. If we have relations *r*1(*R*1) and *r*2(*R*2), then *r*1 (*r*2 is a relation whose schema is the
(A) concatenation (B) union
(C) intersection (D) None

9. Match the following:

I	Empid	1	Multivalued
II	Name	2	Derived
III	Age	3	Composite
IV	Contact No.	4	Simple

(A) I – 4, II – 3, III – 2, Iv – 1
(B) I – 3, II – 2, III – 4, Iv – 1
(C) I – 2, II – 1, III – 4, Iv – 3
(D) I – 1, II – 3, III – 2, IV – 4

10. Match the following:

I	Double-lined ellipse	1	Multivalued attribute
II	Double line	2	Total participation
III	Double-lined box	3	Weak entity set
IV	Dashed ellipse	4	Derived attribute

(A) I – 1, II – 2, III – 3, IV – 4
(B) I – 2, II – 3, III – 4, IV – 1
(C) I – 3, II – 4, III – 2, IV – 2
(D) I – 4, II – 3, III – 2, IV –1

11. The natural join is a
(A) binary operation that allows us to combine certain selections and a Cartesian product into one operation
(B) unary operations that allows only Cartesian product
(C) query which involves a Cartesian product and a projection
(D) None

12. The number of entities participating in the relationship is known as
(A) maximum cardinality (B) composite identifiers
(C) degree (D) None

13 A minimum cardinality of 0 specifies
(A) non-participation
(B) partial participation
(C) total participation
(D) zero participation

14. What is not true about weak entity?
(A) They do not have key attributes.
(B) They are the examples of existence dependency.
(C) Every existence dependency results in a weak entity
(D) Weak entity will have always discriminator attributes

15. Which one is the fundamental operation in the relational algebra?
(A) Natural join (B) Division
(C) Set intersection (D) Cartesian product

16. For the given tables

A

X	Y
a_1	b_1
a_2	b_1
a_1	b_2
a_2	b_2

B

Y
b_1
b_2

$A \div B$ will return
(A) a_1, a_2 (B) a_1
(C) a_2 (D) None

17. The number of tuples selected in the above answer is
(A) 2 (B) 1
(C) 0 (D) 4

Common data for questions 18 and 19: Consider the following schema of a relational database.

Emp (empno, name, add)

Project (Pno, Prame)

Work on (empno, Pno)

Part (partno, Pname, qty, size)

Use (empno, pno, partno, no)

18. ((name(emp) ((name(emp ⋈ workon) displays
(i) The names of the employees who are not working in any project
(ii) The names of the employees who were working in every project.
(A) Only (i) (B) Only (ii)
(C) Both (A) and (B) (D) None

19. List the partno and names of the parts used in both the projects DBMS & MIS:

(A) $\sigma_{partno,\ pname,}{}^{(part} \bowtie {}^{(}\pi_{partno}(\sigma_{pname} = \text{“DBMS”}{}^{(project)} \bowtie {}^{use)}\cap \pi_{partno}(\sigma_{pname} = \text{“MIS”}{}^{(project)} \bowtie {}^{use)})$

(B) (partno, pname(part ⋈ ((partno ((pname = “DBMS”(project ⋈ use) ((partno(σ_{pname} = “MIS”$^{(project)} \bowtie {}^{use)}$)))

(C) $\pi_{partno,\ pname}$(part ⋈ (σ_{partno} (σ_{pname} = “DBMS”$^{(project} \bowtie {}^{use)}$ ((partno ((pname = “MIS”(project) ⋈ use)))

(D) None

20. The following query shows. SELECT job-status, sum (basic – salary) AVG (basic – salary) from employees group by job – status.

(i) It shows job status, sum, AVG of all data

(ii) It shows job status, Sum, AVG, with group by clause in use.

(A) only (i)

(B) only (ii)

(C) both (A) and (B)

(D) None

Previous Years' Questions

1. Let E_1 and E_2 be two entities in an E/R diagram, with simple single-valued attributes. R_1 and R_2 are two relationships between E_1 and E_2, where R_1 is one-to-many and R_2 is many-to-many. R_1 and R_2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relational model? **[2005]**

(A) 2 (B) 3

(C) 4 (D) 5

2. The following table has two attributes A and C where A is the primary key and C is the foreign key referencing A with on-delete cascade.

A	C
2	4
3	4
4	3
5	2
7	2
9	5
6	4

The set of all tuples that must be additionally deleted to preserve referential integrity when the tuple (2, 4) is deleted is: **[2005]**

(A) (3, 4) and (6, 4)

(B) (5, 2) and (7, 2)

(C) (5, 2), (7, 2) and (9, 5)

(D) (3, 4), (4, 3) and (6, 4)

3. Which of the following tuple relational calculus expression(s) is/are equivalent to $\forall t \in r\ (P(t))$?

I. $\neg \exists t \in r\ (P(t))$

II. $\exists t \notin r\ (P(t))$

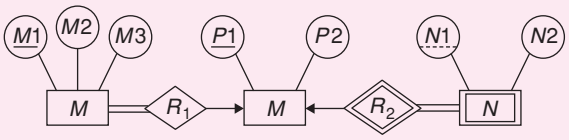
III. $\neg \exists t \in r\ (\neg P(t))$

IV. $\exists t \in r\ (\neg P(t))$ **[2008]**

(A) I only (B) II only

(C) III only (D) III and IV only

Common data for questions 4 and 5: Consider the following ER diagram:



4. The minimum number of tables needed to represent M, N, P, R_1, R_2 is? **[2008]**

(A) 2 (B) 3

(C) 4 (D) 5

5. Which of the following is a correct attribute set for one of the tables for the correct answer to the above question? **[2008]**

(A) {$M1$, $M2$, $M3$, $P1$} (B) {$M1$, $P1$, $N1$, $N2$}

(C) {$M1$, $P1$, $N1$} (D) {$M1$, $P1$}

6. Consider a relational table with a single record for each registered student with the following attributes.

1. *Registration_Num*: Unique registration number of each registered student
2. *UID*: Unique identity number, unique at the national level for each citizen
3. *BankAccount_Num*: Unique account number at the bank. A student can have multiple accounts or joint accounts. This attribute stores the primary account number
4. *Name*: Name of the student
5. *Hostel_Room*: Room number of the hostel

Which of the following options is incorrect? **[2011]**

(A) *BankAccount_Num* is a candidate key

(B) *Registration_Num* can be a primary key

(C) *UID* is a candidate key if all students are from the same country

(D) If S is a super key such that $S \cap UID$ is NULL then $S \cup UID$ is also super key.

7. Given the basic ER and relational models, which of the following is incorrect? **[2012]**

(A) An attribute of an entity can have more than one value

(B) An attribute of an entity can be composite

(C) In a row of a relational table, an attribute can have more than one value

(D) In a row of a relational table, an attribute can have exactly one value or a NULL value

8. An ER model of a database consists of entity types A and B. These are connected by a relationship R which does not have its own attribute, Under which one of the following conditions, can the relational table for R be merged with that of A? **[2017]**

(A) Relationship R is one-to-many and the participation of A in R is total.
(B) Relationship R is one-to-many and the participation of A in R is partial.
(C) Relationship R is many-to-one and the participation of A in R is total.
(D) Relationship R is many-to-one and the participation of A in R is partial.

9. In an Entity-Relationship (ER) model, suppose R is a many-to-one relationship from entity set $E1$ to entity set $E2$. Assume that $E1$ and $E2$ participate totally in R and that the cardinality of $E1$ is greater than the cardinality of $E2$.

Which one of the following is true about R? **[2018]**

(A) Every entity in $E1$ is associated with exactly one entity in $E2$.
(B) Some entity in $E1$ is associated with more than one entity in $E2$.
(C) Every entity in $E2$ is associated with exactly one entity in $E1$.
(D) Every entity in $E2$ is associated with at most one entity in $E1$.

Answer Keys

Exercises

Practice Problems 1

1. (i) B (ii) A (iii) A **2.** A **3.** B **4.** D **5.** D **6.** D **7.** B **8.** B
9. B **10.** A **11.** A **12.** B **13.** D **14.** A **15.** A **16.** C **17.** B **18.** B
19. C **20.** D

Practice Problems 2

1. D **2.** B **3.** C **4.** A **5.** A **6.** D **7.** C **8.** A **9.** A **10.** A
11. A **12.** C **13.** C **14.** A **15.** D **16.** A **17.** A **18.** A **19.** C **20.** B

Previous Years' Questions

1. B **2.** C **3.** D **4.** B **5.** A **6.** A **7.** C **8.** C **9.** A

Chapter 2

Structured Query Language

LEARNING OBJECTIVES

- *Relational algebra*
- *Select operator*
- *Project operator*
- *Set operators*
- *Union compatible relations*
- *Union operation*
- *Aggregate operators*
- *Correlated nested queries*
- *Relational calculus*
- *Tuple relational calculus*
- *Tuple relational calculus*
- *DML*
- *Super key*
- *SQL commands*

Relational Algebra

1. A set of operators (unary or binary) that take relation instances as arguments and return new relations.
2. Gives a procedural method of specifying a retrieval query
3. Forms the core component of a relational query engine
4. *SQL* queries are internally translated into *RA* expressions
5. Provides a framework for query optimization

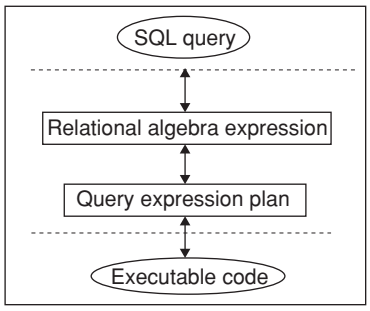
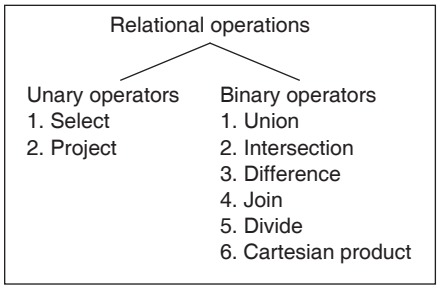


Figure 1 *Role of relational algebra in DBMS:*

Relational Operations

A collection of simple 'low-level' operations used to manipulate relations.

1. It provides a procedural way to query a database.
2. Input is one (or) more relations.
3. Output is one relation.



Select Operator (σ)

Select operator is an unary operator. It can be used to select those tuples of a relation that satisfy a given condition.

Notation: $\sigma_\theta(r)$

σ: Select operator(read as sigma)

θ: Selection condition

r: Relation name

Result is a relation with the same scheme as r consisting of the tuples in r that satisfy condition θ

Syntax: $\sigma_{\text{condition}}$ (relation)

Example:

Table 2.1 *Person*

Id	Name	Address	Hobby
112	John	12, SP Road	Stamp collection
113	John	12, SP Road	Coin collection
114	Mary	16, SP Road	Painting
115	Brat	18, GP Road	Stamp collection

$\sigma_{\text{Hobby = 'stamp. Collection'}}$(person)

The above given statement displays all tuples (or) records with hobby 'stamp collection'.

Output:

Id	Name	Address	Hobby
112	John	12, SP Road	Stamp collection
115	Brat	18, GP Road	Stamp collection

Selection condition can use following operators:
<, ≤, >, ≥, =, ≠

1. <attribute> operator <attribute>
2. <attribute> operator <constant>

Example: Salary ≥ 1000

3. <Condition> AND/OR <condition>

Example: (Experience > 3) AND (Age < 58)

4. NOT <condition>

Selection operation examples:

1. $\sigma_{\text{Id > 112 OR Hobby = 'paint'}}$ (person)
   It displays the tuples whose ID > 112 or Hobby is paint
2. $\sigma_{\text{Id > 112 AND Id < 115}}$ (person)
   It displays tuples whose ID is greater than 112 and less than 115
3. $\sigma_{\text{NOT (hobby = 'paint')}}$ (person)
   It displays tuples whose hobby is not paint
4. $\sigma_{\text{Hobby} \neq \text{'paint'}}$ (person)
   It displays tuples whose hobby is not paint, displays all tuples other than hobby paint.
   Selection operator: Produces table containing subset of rows of argument table which satisfies condition.

Project Operator (π)

The project operator is unary operator. It can be used to keep only the required attributes of a relation instance and throw away others.

Notation: $\pi_{A2, A2, \ldots Ak(r)}$ Where $A_1, A_2, \ldots A_K$ is a list L of desired attributes in the scheme of r.

Result = $\{(V_1, V_2, \ldots V_K)/V_i \in \text{DOM}(A_i), 1 \leq i \leq k$ and there is some tuple t in r, such that $t.A_1 = v_1, t.A_2 = v_2, \ldots t.A_K = V_K\}$
$\pi_{\text{Attribute List}}$ (Relation)

Take table 2.1 as reference.

1. π_{Name} (person)

Output:

Name
John
Mary
Bart

In the output table, John name has appeared once, project operation eliminated duplicates.

2. $\pi_{\text{Name, address}}$ (person)

Output:

Name	Address
John	12, SP Road
Mary	16, SP Road
Bart	18,GP Road

Expressions:

$\pi_{\text{id,name}}$ ($\sigma_{\text{hobby = 'stamp collection' OR Hobby= 'coin collection'}}$(person))

Output:

Id	Name
112	John
115	Bart

The above given relational algebra expression gives Ids, names of a person whose hobby is either stamp collection (or) coin collection.

Set Operators

Union (∪), Intersection (∩), set difference (–) are called *set operators*. Result of combining two relations with a set operator is a relation ⇒ all its elements must be tuples having same structure. Hence scope of set operations is limited to union compatible relations.

Union Compatible Relations

Two relations are union compatible if

1. Both have same number of columns
2. Names of attributes are same
3. Corresponding fields have same type
4. Attributes with the same name in both relations have same domain.
5. Union compatible relations can be combined using Union, Intersection, and set difference.

Example:

Consider the given tables.
Person (SSN, Name, Address, Hobby)
Professor (Id, Name, office, phone)
person and professor tables are not union compatible.

Union

The result of union will be a set consisting of all tuples appearing in either or both of the given relations. Relations cannot contain a mixture of different kinds of tuples, they must be 'tuple – homogeneous'. The union in the relational algebra is not the completely general mathematical union; rather, it is a special kind of union, in which we require the two input relations to be of the same type.

The general definition of relational union operator:

Given are two relations '*a*' and '*b*' of the same type. The union of those two relations, a union *b*, is a relation of the same type, with body consisting of all tuples '*t*' such that '*t*' appears in a or *b* or both.

* Union operation eliminates duplicates.

Here is a different but equivalent definition:

Given are two relations '*a*' and '*b*' of the same type. The union of those two relations, *a* union *b*, is a relation of the same type, with body consisting of all tuples t such that t is equal to (i.e., is a duplicate of) some tuple in *a* or *b* or both.

Union Operation (U)

When union operation is applied on two tables it gives all the tuples in both without Repetition.

Example:

Table 2 *Result of union operation*

R

Roll. no.	Name	Semester	Percentage
22	Arun	7	45%
31	Bindu	6	55%
58	Sita	5	35%

S

Roll no.	Name	Semester	Percentage
28	Suresh	4	65%
31	Bindu	6	55%
44	Pinky	4	75%
58	Sita	5	35%

$R \cup S$

Roll no.	Name	Semester	Percentage
22	Arun	7	45%
31	Bindu	6	55%
58	Sita	5	35%
44	Pinky	4	75%
28	Sita	5	35%

Intersection

Like union, Intersection operator requires its operands to be of the same type. Given are two relations *a* and *b* of the same type, then, the intersection of those two relations, '*a*' INTERSECT '*b*', is a relation of the same type, with body consisting of all tuples t such that t appears in both '*a*' and '*b*'.

Intersection operation returns tuples which are common to both tables

Table 3 *Result of intersection operation*

$R \cap S =$

Roll no.	Name	Semester	Percentage
31	Bindu	6	55%
58	Sita	5	35%

Difference

Like union and intersection, the relational difference operator also requires its operands to be of the same type. Given are two relations '*a*' and '*b*' of the same type, Then, the difference between those two relations, '*a*' MINUS '*b*' (in that order), is a relation of the same type, with body consisting of all types t such that t appears in a and not *b*.

1. MINUS has a directionality to it, just as subtraction does in ordinary arithmetic (e.g., '6 – 3' and '3 – 6' are not the same thing)
2. Redundant duplicate rows are always eliminated from the result of UNION, INTERSECTION, EXCEPT operations.
3. SQL also provides the qualified variants UNION ALL, INTERSECT ALL and EXCEPT ALL, where duplicates are retained

Set difference operation returns the tuples in the first table which are not matching with the tuples of other table.

Table 4 *Result of R – S*

R – S =

Roll no.	Name	Semester	Percentage
22	Arun	7	45%

Table 5 *Result of S – R*

S – R =

Roll no.	Name	Semester	Percentage
28	Suresh	4	65%
44	Pinky	4	75%

* R – S ≠ S –R (both are different)

Example:

A

Supplier number	Supplier name	Status	City
SN1	MAHESH	40	HYDERABAD
SN3	SURESH	40	HYDERABAD

B

Supplier number	Supplier number	Status	City
SN3	SURESH	40	HYDERABAD
SN4	RAMESH	30	CHENNAI

UNION ($A \cup B$)

Supplier number	Supplier name	Status	City
SN1	MAHESH	40	HYDERABAD
SN3	SURESH	40	HYDERABAD
SN4	RAMESH	30	CHENNAI

INTERSECTION ($A \cap B$)

Supplier number	Supplier name	Status	City
SN3	SURESH	40	HYDERABAD

DIFFERENCE ($A - B$)

Supplier name	Supplier name	Status	City
SN1	MAHESH	40	HYDERABAD

DIFFERENCE ($B - A$)

Supplier name	Supplier name	Status	City
SN4	RAMESH	30	CHENNAI

Cartesian Product

The Cartesian product of two sets is the set of all ordered pairs such that in each pair, the first element comes from the first set and the second element comes from second set.

The result consists of all the attributes from both of the two input headings. We define the Cartesian product of two relations '*a*' and '*b*', as

'*a*' times '*b*', where *a* and *b* have no common attribute names (If we need to construct the Cartesian product of two relations that do have any such common attribute names, therefore, we must use the RENAME operator first to rename attributes appropriately).

The Cartesian product operation is also known as CROSS PRODUCT. This is also a binary set operation, but the relations on which it is applied need not to be union compatible. This operation is used to combine tuples from two relations in a combinational fashion.

Example:

R

A	B
X_1	X_2
X_3	X_4

S

C	D
Y_1	Y_2
Y_3	Y_4

$R \times S =$

A	B	C	D
X_1	X_2	Y_1	Y_2
X_1	X_2	Y_3	Y_4
X_3	X_4	Y_1	Y_2
X_3	X_4	Y_3	Y_4

Example: Transcript (StuId, coursecode, semester, grade) Teaching (ProfId, coursecode, semester)

$$\pi_{\text{stuId,coursecode}} (\text{Transcript}) \times \pi_{\text{profId,coursecode}} (\text{Teaching})$$

The above expression returns

Table 6 *Result of cross product*

Stu Id	Course code	Prof Id	Course code
...	...	...	...

Aggregate Operators

SQL Supports the usual aggregate operators COUNT, SUM, AVG, MAX, MIN, EVERY and ANY, but there are a few SQL-specific points.

1. The argument can optionally be preceded by the keyword DISTINCT, for example SUM (DISTINCT column -name) to indicate that duplicates are to be eliminated before the aggregation is done. For MAX, MIN, EVERY and ANY, however, DISTINCT has no effect and should not be specified.
2. The operator COUNT (*), for this DISTINCT is not allowed, and is provided to count all rows in a table without any duplicate elimination.
3. Any NULLS in the argument column are eliminated before the aggregation is done, regardless of whether DISTINCT is specified, except in the case of COUNT (*), where nulls behave as if they were values.
4. After NULLS if any have been eliminated, if what is left is an empty set, COUNT returns zero. The other operators return NULL.

AVG, MIN, MAX, SUM, COUNT

These functions operate on the multiset of values of column of a relation and returns a value.

1. Find the average account balance at the Perryridge branch.

Solution: SELECT AVG (balance) FROM account WHERE branch.name = 'perryridge'

2. Find the number of tuples in customer relation.

Solution: SELECT count (*) FROM customer

3. Find the number of depositors for each branch.

Solution: SELECT branch.name, COUNT (distinct customer-name) FROM depositor, account WHERE depositor. account-no = account account-no GROUPBY branch-name.

Nested Queries

Some queries require that existing values in the database be fetched and then used in a comparison condition.

Such queries can be conveniently formulated by using nested queries, which are complete SELECT – FROM –WHERE blocks within the WHERE clause of another query. The other query is called the *outer query*.

In-Comparison Operator

The comparison operator IN, which compares a value '*v*' with a set of values '*V*' and evaluates to TRUE if '*v*' is one of the elements in *V*.

Example: Consider the given database scheme and the statement:

EMPLOYEE

FNAME	INITIAL	LNAME	ENO	DOB	ADDRESS	SALARY	DNO

DEPARTMENT

D NAME	DNO	MANAGER-NO

DEPARTMENT–LOCATIONS

DNO	D-LOCATION

PROJECT

PNAME	PNO	P-LOCATION	DNO

WORKS–ON

ENO	PNO	HOURS

Example: Select distinct PNO from project where PNO IN (select PNO from project, department, employee where P. DNO = D. DNO AND MANAGER. NO = ENO AND LNAME = 'RAMYA')

The first query selects. The project numbers that have a 'Ramya' involved as manager, while the second selects the project numbers of projects that have a 'Ramya' involved as worker.

If a nested returns a single value, in such cases, it is permissible to use = instead of IN for the comparison operator.

In general, the nested query will return a table, Which is a set of multiset of tuples.

* SQL allows the use of tuples of values in comparisons by placing them within parentheses.

Example: SELECT DISTINCT ENO FROM WORKS - ON WHERE (PNO, HOURS) IN (SELECT PNO, HOURS FROM WORKS - ON WHERE ENO = 929).

This query will select the employee numbers of all employees who work on the same (PROJECT, HOURS) combination on some project a particular employee whose ENO = '929' works on. In this example, the IN operator compares the subtuple of values in parentheses (PNO, HOURS) for each tuple in works on with the set of union-compatible tuples produced by the nested query.

Correlated nested queries: Nested queries can be evaluated by executing the sub query (or) Inner query once and substituting the resulting value (or) values into the WHERE clause of the outer query.

1. In co-related nested queries, the inner query depends on the outer query for its value.
2. Sub-query is executed repeatedly, once for each row that is selected by the outer query.
3. A correlated subquery is a sub query that contains a reference to a table that also appears in the outer query.

Example: Consider the following correlated nested query:

SELECT *
FROM table1
WHERE col1 ≥ ALL
(SELECT col1 FROM table2 WHERE table2. col2 = table1. col2)

1. The subquery contains reference to a column of table1, even though the sub-queries FROM clause does not mention a table table1
2. SQL has to check outside the sub-query and find Table 1 in the outer query
3. Suppose that Table 1 contains a row where col1 = 3 and col2 = 4 and Table 2 contains a row where col1 = 5 and col2 = 4
4. The expression

   WHERE col1 ≥ All (SELECT col1 FROM table2

   3 ≥ 5 (false)

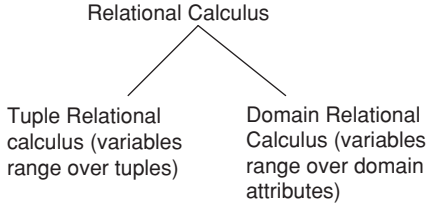
   (WHERE condition TRUE) Table1. col2 = table2. col2 4 = 4

   So the expression as a whole is FALSE.
5. It is evaluated from outside to inside

Relational Calculus

Relational calculus can define the information to be retrieved

1. In this, there is no specific series of operations.
2. Relational algebra defines the sequence of operations.
3. Relational calculus is closer to how users would formulate queries, in terms of information requirements, rather than in terms of operations.



4. Relational calculus is based on predicate logic, gives the usual quantifiers to construct complex queries.

Tuple Relational Calculus

Example: Employee

E Id	F Name	L Name	S alary
201	John	James	3000
202	Brat	Frank	2000
203	Mary	Jennifer	3000
204	Adam	Borg	2000
205	Smith	Joyce	1000

Query 1: Display the employees whose salary is above 2000.

$\{E \mid \exists E \in \text{Employee}(E.\text{salary} > 2000)\}$

Output:

E Id	F Name	L Name	Salary
201	John	James	3000
203	Mary	Jennifer	3000

Query 2: Display the employee Ids whose salary is above 1000 and below 3000.

$\{P \mid \exists E \in \text{Employee}((E.\text{salary} > 1000 \wedge E.\text{salary} < 3000) \wedge P.\text{EId} = E.\text{EId})\}$

P is a table, in which EIds are stored, from tuples which satisfies the given condition.

Tuple Relational Calculus

A non-procedural language, where each query is of the form $\{t \mid p(t)\}$. It is a set of all tuples t such that predicate p is true for t, t is a tuple variable, $t[A]$ denotes the value of tuple 't' on attribute A.

$$\{T \mid p(T)\}$$

T is a tuple and $P(T)$ denotes a formula in which tuple variable T appears.

1. $\forall \times (P(X))$
   $\forall$ is called the universal or 'for all' quantifier because every tuple in 'the universe of' tuples must make F true to make the quantified formula true.
   Only true if $p(X)$ is true for every X in the universe.

   Example: $\forall \times (\text{x.color} = \text{'Red'})$
   means everything that exists is red.

   Example: $\forall \times ((x \in \text{Boats}) \Rightarrow (X.\text{color} = \text{'Red'}))$
   '$\Rightarrow$' is a logical implication. $a \Rightarrow b$ means that if a is true, b must be true
   (or)
   $\exists \times \in \text{Boats}\ (X.\text{color} = \text{'Red'})$
   For every 'x' in the boats relation, the color must be red.

2. $\exists \times (P(X))$
   $\exists$ is called the *existential* or '*there exists*' quantifier because any tuple that exists in 'the universe of' tuples may take F true, to make the quantified formula true.

   Example: $\exists \times ((X \in \text{Boats}) \wedge \text{X.color}) = \text{'Red'})$
   There exists a tuple X in the boats relation whose color is red.
   (or)
   $\exists \times \in \text{Boats}\ (\text{X.color} = \text{'Red'})$

Examples:

1. Find all sailors with rating above 8.

Sailors:

Sid	Sname	Rating	Age
28	Yuppy	9	35
35	Rubber	8	55
44	grove	5	35
58	rusty	10	35

Solution: $\{s \mid s \in \text{sailors} \wedge s.\ \text{rating} > 8\}$

Output

$R =$

Sid	Sname	Rating	Age
28	yuppy	9	35
58	rusty	10	35

2. Find names and ages of sailors with rating > 8.

   Solution: $\{R \mid \exists S \in \text{sailors}\ (\text{s.rating} > 8 \wedge \text{R.sname} = \text{s.name} \wedge \text{R.age} = \text{s.age})$

   Output:

sname	age
yuppy	35
rusty	35

Join Operation in Tuple Relational Calculus

Examples:

3. Find sailors rated > 7 who have reserved boat = 103.

Solution: $\{S \mid S \in \text{sailors} \wedge \text{s.rating} > 7 \wedge \exists R\ (R \in \text{reserves} \wedge \text{R.sid} = \text{s.sid} \wedge \text{R.bid} = 103)\}$

4. Find sailors rated > 7 who have reserved a red boat.

Solution: $\{S \mid S \in \text{sailors} \wedge \text{s.rating} > 7 \wedge \exists R\ (R \in \text{reserves} \wedge \text{R.sid} = \text{s.sid} \wedge \exists B(\text{Boats} \wedge \text{B.bid} = \text{R.bid} \wedge \text{B.color} = \text{'Red'}))\}$

Division Operation in Tuple Relational Calculus

Examples

1. Find sailors who have reserved all boats.

Solution: $\{S \mid S \in \text{sailors} \wedge \forall B \in \text{boats}\ (\exists R \in \text{reserves}\ (\text{s.sid} = \text{R.sid} \wedge \text{B.bid} = \text{R.bid}))\}$

Domain Relational Calculus

1. Tuple relational and domain relational are semantically similar.
2. In *TRC*, tuples share an equal status as variables, and field referencing can be used to select tuple parts.

3. In *DRC* formed variables are explicit.
4. *DRC* query has the following form.
   $\{<x_1, x_2, \ldots x_n> / P(<x_1, x_2, \ldots, x_n>\}$
   Result included all tuples $<x_1, x_2, \ldots x_n>$
   That make the formula $p(<x_1, x_2, \ldots x_n>)$ true.
5. Formula given in *DRC* is recursively defined. First start with simple atomic formula and expand the formulas by using the logical connectives.
6. A variable that is not bound is free.
7. The variable $X_1, X_2, \ldots, X_n$ that appear in the left side of '/' must be the only free variable in the formula $p(\ldots)$.

Example: Consider the employee table given in the above Example.
The use of quantifiers $\exists x$ and $\forall x$ in a formula is said to bind x

Query 1: Display the Employees whose salary is above 2000?

$$\{<I, F, L, S> / <I, F, L, S> \in \text{Employee} \wedge S > 2000\}$$

Query 2: Display the EIds of Employees, whose salary is above 1000 and below 3000?

$$\{<I> / \exists F, L, S(<I, F, L, S> \in \text{Employee} \wedge (S > 1000 \wedge S < 3000))\}$$

SQL (Structured Query Language)

When a user wants to get some information from a database file, he/she can issue a query. A query is a user-request to retrieve data (or) information with a certain condition. SQL is a query language that allows user to specify the conditions (instead of algorithms)

Concept of SQL

The user specifies a certain condition. The program will go through all the records in the database file and select those records that satisfy the condition. The result of the query will be stored in the form of a table.

Features of SQL

1. SQL is a language of database. It includes database creation, deletion, fetching rows and modifying rows.
2. SQL is a structured query language for storing, manipulating and retrieving data stored in relational database.
3. It allows users to describe the data.
4. It allows users to create and drop database and tables.
5. It allows users to create view, functions in a database.
6. Allows users to set permissions on tables and views.
7. The standard SQL commands to interact with relational database are CREATE, SELECT, UPDATE, INSERT, DROP and DELETE.
8. The commands can be classified as follows:
   - *Data query language*: SELECT – It retrieves particular rows which satisfies the given condition.
   - *Data definition language*: CREATE, ALTER, DROP.
   - *Data manipulation language*: INSERT, UPDATE, DELETE

Features

1. Strong data protection
2. Robust transactional support
3. High performance
4. High availability
5. Security and flexibility to run anything
6. Easy to manage
7. User friendly

General Structure

SELECT ... FROM ... WHERE
SQL is divided into two languages

1. DML (data manipulation language)
   SELECT: Extracts data from a database table.
   UPDATE: Updates data in a database table.
   DELETE: Deletes data from a database table.
   INSERT INTO: Inserts new data into database table.
2. DDL (data definition language)
   CREATE TABLE - creates a new database table.
   ALTER TABLE - Alters a database table.
   DROP TABLE - deletes a database table.
   CREATE INDEX - Creates an index (search key).
   DROP INDEX - Deletes an index.
   RENAME – Changes the name of the table.

Types of keys:

1. Candidate key
2. Primary key
3. Super key
4. Foreign key
5. Composite primary key

In relational database, 'keys' play a major role. Keys are used to establish and identify relation between relations (or) tables.

Keys are used to ensure that each record within a table can be uniquely identified by combining one or more fields (or) column headers within a table.

Candidate key: A candidate key is a column or set of columns in a table that contains unique values, with these we can uniquely identify any database record without referring to any other columns data.

Each table may have one or more candidate keys, among the available candidate keys, one key is preserved for primary key.

A candidate key is a subset of a super key.

Example: Student

StudentId	First name	Last name	Course Id
*C*S00345	Jim	Black	*C*2
*C*S00254	Carry	Norris	*C*1
*C*S00349	Peter	Murray	*C*1
*C*S00196	John	Mc Cloud	*C*3
*C*S00489	Brat	Holland	*C*4
*C*S00553	Mary	Smith	*C*5

In the above table, we have studentId that uniquely identifies the students in a student table. This would be a candidate key.

In the same table, we have student's first name and last name, which are also candidate keys.

1. If we combine first name and last name then also it becomes a candidate key.
2. Candidate key must (have)
   - Unique values
   - No null values
   - Minimum number of fields to ensure uniqueness.
   - Uniquely identify each record in the table.
3. The candidate keys which are not selected for primary key are known as secondary keys or alternative keys.

Primary key: A primary key is a candidate key that is most suitable (or) appropriate to become main key of the table.

1. It is a special relational database table column ((or) combination of columns)
2. Primary key main features are
   - It must contain a unique value for each row of data.
   - It cannot contain null values.

Example: We can choose primary key as studentId which is mentioned in the table given in above example.

Composite primary key: A key that consists of two or more attributes that uniquely identify an entity is called *composite key* or *composite primary key.*

Example: Customer

Cust-Id	Order-Id	Sale-details
*C*1	*O* – 2	Sold
*C*1	*O* – 3	Sold
*C*2	*O* – 2	Sold
*C*2	*O* – 3	Sold

Composite primary key is {cust-Id, order-Id}

Super key: A super key is a combination of attributes that can be uniquely used to identify a database record. A table can have any number of super keys.

1. Candidate key is a special subset of super keys.

Example: Customer

Customer name	Customer Id	SSN	Address	DOB

Assume that we can guarantee uniqueness only for SSN field, then the following are some of the super keys possible.

1. {Name, SSN, DOB}
2. {ID, Name, SSN}

In a set of attributes, there must be at least one key (could be primary key or candidate key)

Foreign key: A foreign key is a column or group of columns in a relational database table that provides connectivity between data in two tables.

1. The majority of tables in a relational database system adhere to the concept of foreign key.
2. In complex databases, data must be added across multiple tables, thus the link or connectivity has to be maintained among the tables.
3. The concept of Referential Integrity constraint is derived from Foreign key.

Example: Emp

EId	EName	Dept – No

Dept

Dept-No	DName

In the above specified tables, Dept-No is common to both the tables, In Dept table it is called as primary key and in Emp table it is called as *foreign key*.

These two tables are connected with the help of 'Dept-No' field

1. For any column acting as a foreign key, a corresponding value should exist in the link (or) connecting table.
2. While inserting data and removing data from the foreign key column, a small incorrect insertion or deletion destroys the relationship between the two tables.

SQL Commands

SELECT statement

The most commonly used SQL command is SELECT statement. The SQL SELECT statement is used to query or retrieve data from a table in the database. A query may retrieve information from specified columns or from all of the columns in the table. To create a simple SQL SELECT statement, you must specify the column(s) names and the table name.

Syntax: SELECT column-name (s) from table name

Example: Persons

Lastname	Firstname	Address	City
Hansen	Ola	SpRoad,-20	Hyd
Svendson	Tove	GPRoad,-18	Secbad
Petterson	Kari	RpRoad,-19	Delhi

1. SELECT lastname FROM persons

   Output:

Lastname
Hansen
Svendson
Petterson

2. SELECT lastname, firstname FROM persons

Output:

Lastname	Firstname
Hansen	Ola
Svendson	Tove
Petterson	Kari

DISTINCT statement

Returns distinct values. It eliminates duplicate values.

Syntax: Select DISTINCT column_name (s) from table-name

Example: Orders

Company	Order.No
IBM	3412
DELL	5614
WIPRO	4412
DELL	4413

1. SELECT company FROM orders

Output:

Company
IBM
DELL
WIPRO
DELL

2. SELECT DISTINCT company FROM orders

Company
IBM
DELL
WIPRO

WHERE statement

The WHERE clause is used when you want to retrieve specific information from a table excluding other irrelevant data. By using WHERE clause, we can restrict the data that is retrieved. The condition provided in the WHERE clause filters the rows retrieved from the table and gives only those rows which were expected. WHERE clause can be used along with SELECT, DELETE, UPDATE statements.

The WHERE clause is used to specify a selection condition. All conditions are specified in this clause.

Syntax: SELECT column FROM table WHERE column operator value.

Operates used in where clause:

=
< > (not equal) (or) ! =
>
<
> =
< =

BETWEEN - Between an inclusive range.
LIKE - Search for a pattern

Example: Persons

Lastname	Firstname	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956
Svendson	Tiva	GPRoad, 18	Sec	1977
Smith	Ole	RPRoad, 19	Hyd	1986
Petterson	Kari	SPRoad, 17	Sec	1985

1. SELECT * FROM persons
Output: It displays the entire table

2. SELECT * FROM persons WHERE city = 'Hyd'

Output:

Lastname	Firstname	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956
Smith	Ole	RPRoad, 19	Hyd	1986

LIKE condition

The LIKE operator is used to list all rows in a table whose column values match a specified pattern. It is useful when you want to search rows to match a specific pattern, or when you do not know the entire value. For this purpose, we use a wildcard character '%'.

The LIKE condition is used to specify a search for a pattern in a column.

A '%' sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.

Syntax: SELECT column FROM table WHERE column LIKE pattern

1. SELECT * FROM persons WHERE Firstname LIKE 'O%'

Solution: SQL statement will return persons with first names that start with a letter '*O*'

Output:

Lastname	Firstname	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956
Smith	Ole	RPRoad, 19	Hyd	1986

2. SELECT * FROM persons WHERE Firstname LIKE '%a'

Solution: SQL statement will return persons whose first name ends with letter 'a'.

Output:

Last name	First name	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956
Svendson	Tiva	GPRoad, 18	Sec. bad	1977

3. SELECT * FROM persons WHERE firstname LIKE '%la%'

Solution: SQL statement returns persons whose firstname contains 'la'. The word sequence 'la' may come at any place in the word.

Output:

Last name	First Name	Address	City	Year
Hansen	Ola	SPRoad, 16	Hyd	1956

String operations

1. '%idge%' matches 'Rockridge', 'Ridgeway', 'Perryridge'.
2. '_____' matches a string of three characters.
3. '_____%' matches a string of at least rhree characters.

INSERT INTO statement

This statement is used to insert new rows into a table. While inserting a row, if you are adding values for all the columns of the table you need not specify the column(s) name in the SQL query. But you need to make sure the order of the values is in the same order as the columns in the table. When adding a row, only the characters or data values should be enclosed with single quotes and ensure the data type of the value and the column matches. One can specify the columns for which you want to insert data

Syntax: INSERT INTO table-name (column1, column2 …) VALUES (value 1, value2 …)

1. INSERT INTO persons VALUES ('Hetland', 'Camilla', 'HPRoad 20', 'Hyd')

Output:

Last name	First Name	Address	City
Hansen	Ola	S.P Road 16	Hyd
Svesdon	Tiva	GP Road 18	Secbad
Smith	Ole	RP Road 19	Hyd
Petterson	Kari	SP Road 17	Secbad
Hetlan	Camilla	HPRoad, 20	Hyd

2. Insert data into specified columns
   INSERT INTO persons (Lastname, Address) VALUES ('Rasmussen', 'street 67')

Output:

Last name	First Name	Address	City
Hansen	Ola	SP Road 16	Hyd
Svesdon	.Tiva	GP Road 18	Secbad
Smith	Ole	RP Road 19	Hyd
Petterson	Kari	SP Road 17	Secbad
Hetlan	Camilla	HP Road 20	Hyd
Rasmussen		Street 67	

UPDATE

The update statement is used to modify the data in a table.

Syntax: UPDATE table_name
SET Column_name = new_value
WHERE column_name = some_value.

1. Add a first name (Nine) to the person whose last name is 'Rasmussen'?

Solution: UPDATE person SET Firstname = 'Nine'
WHERE Lastname = 'Rasmussen'

2. Change the address and add the name of the city as Hyd of a person with last name Rasmussen?

Solution: UPDATE person
SET Address = 'street 12',
city = 'Hyd'
WHERE Lastname = 'Rasmussen'

DELETE statement

The DELETE statement is used to delete rows from a table. The WHERE clause in the SQL delete command is optional, and it identifies the rows in the column that gets deleted. If you do not include the WHERE clause, all the rows in the table will be deleted.

Syntax: DELETE FROM table_name
WHERE column_name = some_value

1. Delete all rows?

Solution: DELETE * FROM table_name

Cartesian product

The Cartesian product of two sets is the set of all ordered pairs of elements such that the first element in each pair belongs to the first set and the second element in each pair belongs to the second set. It is denoted by cross(X).

For example, given two sets:

$S1 = \{1, 2, 3\}$ and $S2 = \{4, 5, 6\}$

The Cartesian product $S1 \times S2$ is the set

$\{(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)\}$

Example:

Female	
Name	Job
Komal	Clerk
Ankita	Sales

Male	
Name	Job
Rohit	Clerk
Raju	Sales

Assume that the tables refer to male and female staff, respectively. Now, in order to obtain all possible inter-staff marriages, the cartesian product can be taken.

Male-Female

Female name	Female job	Male name	Male job
Komal	Clerk	Rohit	Clerk
Komal	Clerk	Raju	Sales
Ankita	Sales	Rohit	Clerk
Ankita	Sales	Raju	Sales

Examples:

1. Find the Cartesian product of borrower and loan?

Solution: SELECT * FROM borrower, loan

2. Find the name, loan-no, and loan amount of all customers having a loan at the Perryridge branch?

Solution: SELECT customer_name, borrower. loan_number, amount FROM borrower, loan WHERE borrower. loan-no = Loan.loan_no AND branch_name = 'perryridge'

3. Find all loan numbers for loans made at the perryridge branch with loan amount greater than 1200?

Solution: SELECT loan-no FROM loan WHERE branch. name = 'perryridge' AND amount > 1200

Comparison operator

Relation algebra includes six comparison operators (=, < >, <, >, < =, > =). These are proposition forming operators on terms. For example, $x <> 0$ asserts that x is not equal to 0. It also includes three logical operators (AND, OR, NOT). These are proposition forming operators on propositions.

Example: $x > 0$ and $x < 8$

Comparison results can be combined using the logical connections AND, OR NOT

1. Find the loan-no of those loans with amounts between 90,000 and 1,00,000?

Solution: SELECT loan-no FORM loan WHERE amount BETWEEN 90,000 AND 1,00,000. SQL allows renewing relations and attributes using 'AS' clause

2. Find the name, loan-no and loan amount of all customers, rename the column name loan-no as loan.id?

Solution: SELECT customer.name, borrower.loan no AS loan.id, amount FROM borrower, loan, WHERE borrower. loan-no = loan.loan-no

Ordering of Tuples

It lists the tuples in alphabetical order.

Example: List in alphabetic order, the names of all customers having a loan in Perryridge branch?

Solution: SELECT customer-name FROM borrower WHERE branch.name = 'perryridge' ORDERBY customer-name

We may specify 'desc' for descending order (or) 'asc' for ascending order. - 'asc' is default.

Example: ORDERBY customer-name desc.

Join (⋈)

SQL Join is used to get data from two (or) more tables, which appear as single table after joining.

1. Join is used for combining columns from two or more tables by using values common to both tables.
2. Self Join: A table can also join to itself is known as self join. Types of JOIN
   1. INNER JOIN
   2. OUTER JOIN
      (i) LEFT OUTER JOIN
      (ii) RIGHT OUTER JOIN
      (iii) FULL OUTER JOIN

1. INNER JOIN (or) EQUI JOIN

It is a simple JOIN in which result is based on matching tuple, depending on the equality condition specified in the query.

Syntax: SELECT Column-names FROM table name1 INNER JOIN table name 2 WHERE table name 1. Column name = table name 2.column – name.

Example: Class

SID	Name
11	Ana
12	Bala
13	Sudha
14	adam

Info

SID	City
11	Bangalore
12	Delhi
13	Hyderabad

SELECT *
FROM Class INNER JOIN Info
WHERE Class.SID = Info.SID

Result:

SID	Name	SID	City
11	Ana	11	Banglore
12	Bala	12	Delhi
13	Sudha	13	Hyderabad

NATURAL JOIN:

NATURAL JOIN is a type of INNER JOIN which is based on column having same name and same data type present in two tables on which join is performed.

Syntax: SELECT *
FROM table-name1 NATURAL JOIN table-name 2

Example: Consider the tables class and Info, and the following Query

SELECT *
FROM class NATURAL JOIN Info

Result:

SID	Name	City
11	Ana	Bangalore
12	Bala	Delhi
13	Sudha	Hyderabad

Both tables being joined have SID column (same name and same data type), the tuples for which value of SID matches in both the tables, appear in the result.

Dangling tuple: When NATURAL JOIN is performed on two tables, there would be some missing tuples in the result of NATURAL JOIN

Those missing tuples are called *Dangling tuples*. In the above example, the number of dangling tuples is 1 that is

14	Adam

OUTER JOIN: Outer Join is based on both matched and unmatched data.

LEFT OUTER JOIN: Left outer Join returns the tuples available in the left side table with the matched data of 2 tables and null for the right tables column.

Example: Consider the table's class and Info

SELECT *
FROM class LEFT OUTER JOIN Info
ON(class.SID = Info. SID)

Result:

SID	Name	City
11	Ana	Banglore
12	Bala	Delhi
13	Sudha	Hyderabad
14	adam	NULL

RIGHT OUTER JOIN: RIGHT OUTER JOIN returns the tuples available in the Right side table with the matched data of 2 tables and NULL for the left table's column.

Example: Class 1

SID	Name
16	Arun
17	Kamal

Info 1

SID	City
16	Chennai
17	Noida

Query:
SELECT *
FROM Class1 RIGHT OUTER JOIN Info1
ON(class1.SID = Info1.SID)

Result:

SID	Name	City
16	Arun	Chennai
18	NULL	Noida

FULL OUTER JOIN: The full outer Join returns the tuples with the matched data of two tables, remaining rows of both left table and Right table are also included.

Example: Consider the tables class 1 and Info1

Query:
SELECT *
FROM class1 FULL OUER JOIN Info1
ON(class1.SID = Info1.SID)

Result:

SID	Name	City
16	Arun	Chennai
17	Kamal	NULL
18	NULL	Noida

ALTER command: ALTER command is used for altering the table structure

1. It is used to add a new column to existing table.
2. To rename existing column.
3. ALTER is used to drop a column.
4. It is used to change data type of any column or modify its size.

Add new column: By using alter command, we can add a new column to the table.

Syntax: ALTER table table-name ADD(column-name data type).

Example: Consider a student table.

SID	S Name	Grade

Add a new column called address

```
ALTER table student ADD (address char);
```

Example: Add multiple columns, parent-name, course-Name, date-of-birth to student table.

```
ALTER table student ADD (parent-name
varchar(60), course-Name varchar(20),
date-of-birth date);
```

Example: Change the data type of column address to varchar?

```
ALTER table student modify(address varchar(30))
```

Example: Rename a column address to Location

```
ALTER table student rename address to Location
```

TRUNCATE command: Truncate command removes all tuples from a table, this command will not destroy the tables structure.

Syntax: Truncate table table-name

DROP Command: DROP query removes a table completely from database. This command will destroy the table structure.

Syntax: Drop table table-name

Rename: This command is used to rename a table.

Syntax: Rename table old-table-name to new-table-name.

Example: Rename table Employee to New-Employee.

DROP a column: Alter command can be combined with DROP command to remove columns from a table.

Syntax: alter table table-name DROP(column-name)

Example: Alter table student DROP (grade)

EXERCISES

Practice Problems I

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Consider the given table called *Persons*

P-Id	Lastname	Firstname	Address	City
1	Hansen	ola	Timoteivn -10	Sandnes
2	Svendson	Tove	Brazil-50	Sandnes
3	Petterson	Kari	Storgt-20	Stavanger
4	Joseph	ole	Brazil-20	Sandnes

Write a query to select the persons with first name 'Tove' and last name 'Svendson'?

(A)
```
SELECT *
FROM Persons
WHERE first-name='tove'
AND last-name='svendson'
```
(B)
```
SELECT *
FROM Persons
WHERE first-name='tove'
OR last-name='svendson'
```
(C)
```
SELECT first-name
FROM Persons
WHERE first-name='tove'
AND last-name='svendson'
```
(D)
```
SELECT last-name
FROM Persons
WHERE first-name='tove'
AND last-name='svendson'
```

2. Write a query to select only the persons with last name 'Svendson' and the first name equal to 'Tove' or 'ola'?

(A)
```
SELECT *
FROM Persons
WHERE last-name='svendson'
AND first-name='tove'
```
(B)
```
SELECT *
FROM Persons
WHERE  last-name='svendson'
AND (first-name='tove' OR first-name='ola')
```
(C)
```
SELECT *
FROM Persons
WHERE  last-name='svendson'
AND (first-name='tove' AND first-name='ola')
```
(D)
```
SELECT *
FROM Persons
WHERE  last-name='svendson'
OR (first-name='tove' AND first-name='ola')
```

3. Write an SQL statement to add a new row, but only in specified columns, for the persons table add data into columns 'P-Id', 'Last name' and the 'First name' with values (5, Teja, Jakob)?

(A) INSERT INTO Persons VALUES(5,'teja','jakob')
(B) INSERT INTO Persons VALUES(5,teja,jakob)
(C) INSERT INTO Persons (P-Id, last-name, first-name) VALUES(5,'teja','jakob')
(D) INSERT INTO Persons(P-Id, last-name, first-name) VALUES(5,teja,jakob)

4. Write an SQL statement:

(i) To select the persons living in a city that starts with 'S' from the 'Persons' table?

(A)
```
SELECT *
FROM Persons
WHERE  city LIKE 's__'.
```
(B)
```
SELECT *
FROM Persons
WHERE  city LIKE 's%'.
```
(C)
```
SELECT *
FROM Persons
WHERE  city LIKE '%s'.
```
(D)
```
SELECT *
FROM Persons
WHERE city LIKE '_s%'.
```

(ii) To select the persons living in a city that contains the pattern 'tav' from 'Persons' table?

(A)
```
SELECT *
FROM Persons
WHERE city LIKE '_tav_'.
```
(B)
```
SELECT *
FROM Persons
WHERE city LIKE '_tav%'.
```

(C) SELECT *
FROM Persons
WHERE city LIKE '%tav_'.

(D) SELECT *
FROM Persons
WHERE city LIKE '%tav%'.

(iii) To select the persons whose last name starts with '*b*' or '*s*' or '*p*' ?

(A) SELECT *
FROM Persons
WHERE last-name LIKE '*b-s-p*'

(B) SELECT *
FROM Persons
WHERE last-name LIKE '*b%s%p*'

(C) SELECT *
FROM Persons
WHERE last-name LIKE '*b%s%p%*'

(D) SELECT *
FROM Persons
WHERE last-name LIKE '[*bsp*]%'

5. Consider the given table called 'Persons'

P-Id	Last-name	First-name	Address	City
1	Hansen	ola	Timoteivn-10	Sandnes
2	Svendson	Tove	Brazil-50	Sandnes
3	Petterson	Kari	Storgt-20	Stavanger

and the 'Orders' table

O-Id	Order No	P-Id
11	77895	3
12	44678	3
13	22456	1
14	24562	1
15	34764	5

perform NATURAL JOIN operation on both the tables and what is are the O_Id's displayed in the result?

(A) 11, 12, 13 (B) 11, 13, 14
(C) 11, 12, 13, 14 (D) 12, 13, 14

6. Write an SQL to perform FULL JOIN operation on both 'Person' and 'Orders' tables and What is the number of tuples in the Result?

(A) 4 (B) 5
(C) 6 (D) 7

7. Consider the given table 'Result'.

Student Name	Marks
A	55
B	90
C	40
D	80
E	85
F	95
G	82

(i) Find out the students who have scored more than 80 marks, and display them in descending order according to their marks?

(A) SELECT student-name,marks
FROM Result
WHERE marks > 80
ORDERBY marks DESC

(B) SELECT *
FROM Result
WHERE marks > 80
ORDERBY marks DESC

(C) SELECT student-name,marks
FROM Result
WHERE marks > 80
ORDERBY marks

(D) (A) and (B)

(ii) From the above table, find out the top-most three students.

(A) SELECT student-name
FROM Result
ORDERBY marks DESC > 3

(B) SELECT student-name
FROM Result
ORDERBY marks DESC = 3

(C) SELECT student-name
FROM Result
ORDERBY marks DESC limit 3

(D) None of these

8. From the table 'Results', Identify the suitable SQL expression?

(i) Find out the student Who stood 2nd?

(A) SELECT student-name
FROM Result
ORDERBY marks DESC limit 2

(B) SELECT student-name
FROM Result
ORDERBY marks DESC limit 1,1

(C) SELECT student-name
FROM Result
ORDERBY marks DESC limit 1,2

(D) SELECT student-name
FROM Result
ORDERBY marks DESC limit 2,1

(ii) Find out how many students scored > = 80.

(A) SELECT COUNT(*)
FROM Result
WHERE marks > = 80

(B) SELECT COUNT
FROM Result
WHERE marks > = 80

(C) SELECT SUM(*)
FROM Result
WHERE marks > = 80

(D) SELECT SUM
FROM Result
WHERE marks > = 80

9. Consider the given tables:

Customer

Customer name	Customer street	Customer city
Sonam	Mirpurroad	Dhaka
Sonam	Aga KhaRoad	Bogra
Anusha	XYZRoad	Kanchi
Nandy	MirpurRoad	Dhaka

Account

Account number	Customer name	Balance
A-101	Anusha	1000
A-102	Anusha	1500
A-103	Sonam	2000
A-104	Nandy	2500

From the customer table, find out the names of all the customers who live in either Dhaka or Bogra?

(A) SELECT customer-name
FROM customer
WHERE customer-city='dhaka' OR customer-city='bogra'

(B) SELECT customer-name
FROM customer
WHERE customer-city=dhaka OR customer-city='bogra'

(C) SELECT customer-name
FROM customer
WHERE customer-city='dhaka' AND customer-city='bogra'

(D) SELECT customer-name
FROM customer
WHERE customer-city='dhaka' EXIST customer-city='bogra'

10. Consider the given tables

Loan

Loan Number	Branch Name	Amount
L-101	Dhaka	1000
L-103	Khulna	2000

Borrower:

Customer name	Loan number
Sonam	L-101
Nandy	L-103
Anusha	L-103

(i) What are the number of tuples present in the result of cross product of the above two tables?

(A) 4 (B) 5
(C) 6 (D) 7

(ii) Find the loan-numbers from loan table where branch-name is Dhaka?

(A) SELECT loan-number
FROM loan
WHERE branch-name='dhaka'

(B) SELECT loan-number
FROM branch-name='dhaka'

(C) SELECT loan-number
FROM Loan × Borrower

(D) Both (A) and (C)

11. (i) Find all customers who have only accounts but no loans.

(A) SELECT customer-name
FROM depositor LEFT OUTER JOIN Borrower ON
Depositor.customer-name=Borrower.customer-name
WHERE loan-number IS NULL

(B) SELECT customer-name
FROM depositor LEFT OUTER JOIN Borrower ON
Depositor.customer-name = Borrower.customer-name
WHERE loan-number=NULL

(C) SELECT customer-name
FROM depositor RIGHT OUTER JOIN Borrower ON
Depositor.customer-name=Borrower.customer-name
WHERE loan-number IS NULL

(D) SELECT customer-name
FROM depositor RIGHT OUTER JOIN Borrower ON
Depositor.customer-name=Borrower.customer-name
WHERE loan-number=NULL

(ii) Find the names of all customers who have either an account or loan but not both.

Borrower

Customer name	Loan no.
Sonam	L-101
Sonam	L-102
Anusha	L-103

Depositor

Customer name	Account no.
Anusha	A-102
Sonam	A-103
Nandy	A-104

(A) SELECT customer name
FROM depositor FULL OUTER JOIN
Borrower ON
Depositor.customer-name=Borrower.customer-name
WHERE loan-number IS NULL OR Account-number=NULL

(B) SELECT customer-name
FROM depositor FULL OUTER JOIN
Borrower ON
Depositor.customer-name = Borrower.customer-name
WHERE loan-number IS NULL OR Account-number IS NULL

(C) SELECT customer-name
FROM depositor FULL OUTER JOIN
Borrower ON
Depositor.customer-name = Borrower.customer-name
WHERE loan-number = NULL OR Account-number = NULL

(D) SELECT customer-name
FROM depositor FULL OUTER JOIN Borrower ON
Depositor.customer-name = Borrower.customer-name
WHERE loan-number=NULL OR Account-number IS NULL

12. Consider the following 'employee' table

Employee name	Branch name	Branch city	Salary
A	DU	Dhaka	1000
B	DU	Dhaka	2000
C	BUET	Dhaka	3000
D	KUET	Khulna	4000
E	KU	Khulna	5000
F	RU	Rajshahi	6000

(i) Find the distinct number of branches appearing in the employee relation.

(A) SELECT COUNT(branch-name)
FROM Employee

(B) SELECT COUNT(DISTINCT branch-name)
FROM Employee

(C) SELECT DISTINCT COUNT(branch-name)
FROM Employee

(D) SELECT COUNT(*)
FROM Employee

(ii) Find the total salary of all employees at each branch of the bank.

(A) SELECT branch-name, SUM(salary)
FROM Employee
GROUP BY Branch-city

(B) SELECT branch-name, SUM(salary)
FROM Employee
GROUP BY Branch-name

(C) SELECT SUM(salary)
FROM Employee
GROUP BY Branch-name

(D) SELECT branch-name, SUM(salary)
FROM Employee

(iii) Find branch city, branch name Wise total salary, average salary and also number of employees.

(A) SELECT branch-city, branch-name,
SUM (salary), AVG(salary),
COUNT (Employee-name)
FROM Employee
GROUP BY branch-city, branch-name

(B) SELECT branch-city, branch-name,
SUM (salary), AVG (salary),
COUNT (Employee-name)
FROM Employee
GROUP BY branch-city

(C) SELECT branch-city, branch-name,
SUM (salary), AVG (salary), COUNT (Employee-name)
FROM Employee
GROUP BY branch-name

(D) SELECT branch-name, SUM (salary),
AVG (salary), COUNT (Employee-name)
FROM Employee
GROUP BY branch-city, branch-name

Common data for questions 13 to 15: Consider the SHIPMENTS relation and write the SQL statements for the below

SUPPLIERS

Supplier number	Supplier name	Status	City
SN1	Suma	30	Hyderabad
SN2	Hari	20	Chennai
SN3	Anu	10	Hyderabad
SN4	Mahesh	20	Bombay
SN5	Kamal	30	Delhi

PARTS

Part number	Part name	Color	Weight	City
PN1	X	Red	13.0	Chennai
PN2	Y	Green	13.5	Bombay
PN3	X	Yellow	13.2	Hyderabad
PN4	Y	Green	14.1	Calcutta
PN5	Z	Red	14.3	Hyderabad
PN6	Z	Blue	14.2	Bombay

PROJECT

Project number	Project name	City
PJ1	Display	Chennai
PJ2	OCR	Bombay
PJ3	RAID	Chennai
PJ4	SORTER	Hyderabad
PJ5	EDS	Chennai
PJ6	Tape	Bombay
PJ7	Console	Hyderabad

SHIPMENTS

Supplier number	Part number	Project number	Quantity
SN1	PN1	PJ1	300
SN1	PN1	PJ4	400
SN2	PN3	PJ1	350
SN2	PN3	PJ2	450
SN2	PN3	PJ3	640
SN2	PN3	PJ4	320
SN2	PN3	PJ5	330
SN2	PN3	PJ6	520
SN2	PN3	PJ7	480
SN2	PN5	PJ2	460
SN3	PN3	PJ1	440
SN3	PN4	PJ2	410
SN4	PN6	PJ3	310
SN4	PN6	PJ7	320
SN5	PN2	PJ2	340
SN5	PN2	PJ4	350
SN5	PN5	PJ5	360
SN5	PN5	PJ7	370
SN5	PN6	PJ2	380
SN5	PN1	PJ4	420
SN5	PN3	PJ4	440
SN5	PN4	PJ4	450
SN5	PN5	PJ4	400
SN5	PN6	PJ4	410

13. (i) For each part supplied, get the part number and the total shipment quantity?

(A) SELECT shipments.part-number, SUM (shipments.quantity)
FROM Shipments
GROUP BY shipments.part-number

(B) SELECT SUM(shipments.quantity)
FROM Shipments
GROUP BY shipments.part-number

(C) SELECT shipments.part-number, SUM (shipments.quantity)
FROM Shipments
GROUP BY shipments.quantity

(D) SELECT shipments.part-number, SUM (shipments. part-number)
FROM Shipments
GROUP BY shipments.part-number

(ii) Get part numbers for parts supplied by more than two suppliers?

(A) SELECT shipments.part-number
FROM Shipments
GROUP BY shipments.part-number
HAVING COUNT(shipments.supplier-number) > 2

(B) SELECT shipments.part-number
FROM Shipments
GROUP BY shipments.part-number
HAVING COUNT(shipments.supplier-number)>=2

(C) SELECT shipments.part-number
FROM Shipments
GROUP BY shipments.part-number>2

(D) SELECT shipments.part-number, COUNT (shipments.supplier-number)>2
FROM Shipments
GROUP BY shipments.part-number

(iii) Get supplier names for suppliers who supply part PN3?

(A) SELECT DISTINCT suppliers.supplier-name
FROM Supplier
WHERE suppliers.supplier-number IN (SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number='PN3')

(B) SELECT DISTINCT suppliers.supplier-name
FROM Supplier
WHERE suppliers.supplier-number NOT IN(SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number='PN3')

(C) SELECT DISTINCT suppliers.supplier-name
FROM Supplier
WHERE suppliers.supplier-number EXCEPT (SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number='PN3')

(D) SELECT DISTINCT suppliers, supplier-name
FROM Supplier
WHERE suppliers.supplier-number
UNION
SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number='PN3'

14. (i) Get supplier names for suppliers who supply at least one blue part.

(A) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE suppliers.supplier-number
IN (SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number
IN (SELECT Parts.part-number
FROM Parts
WHERE Parts.color='Blue'))

(B) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE suppliers.supplier-number
IN (SELECT Shipments.supplier-number
FROM Shipments

WHERE Shipments.part-number NOT
IN(SELECT Parts.part-number
FROM Parts
WHERE Parts.color='Blue'))

(C) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE suppliers.supplier-number NOT
IN(SELECT Shipments.supplier-number
FROM Shipments
WHERE Shipments.part-number
IN (SELECT Parts.part-number
FROM Parts
WHERE Parts.color='Blue'))

(D) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE suppliers.supplier-number
IN (SELECT Shipments.supplier-name
FROM Shipments
WHERE Shipments.part-number
IN (SELECT Parts.part-number
FROM Parts
WHERE Parts.color='Blue'))

(ii) Get supplier numbers for suppliers with status less than the current maximum status in the suppliers table:

(A) SELECT Suppliers.supplier-number
FROM suppliers
WHERE Suppliers.status < (SELECT MAX
(Suppliers.status)
FROM Suppliers)

(B) SELECT Suppliers.supplier-number
FROM suppliers
WHERE Suppliers.status<=(SELECT MAX
(Suppliers.status)
FROM Suppliers)

(C) SELECT Suppliers.supplier-number,
MAX (Suppliers.status)
FROM suppliers
WHERE Suppliers.status

(D) SELECT Suppliers.supplier-number
FROM suppliers
WHERE Suppliers.status=MAX(Suppliers.
status)

(iii) Get supplier names for suppliers who supply part PN2?

(A) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(B) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(C) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
OR
Shipments.part-number='PN2')

(D) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
UNION
Shipments.part-number='PN2')

15. (i) Get supplier names for suppliers who do not supply part PN2.

(A) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(B) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(C) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXCEPT(SELECT *
FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number='PN2')

(D) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Shipments

WHERE Shipments.supplier-number = suppliers.supplier-number
OR
Shipments.part-number='PN2')

(ii) Get supplier names for suppliers who supply all parts.

(A) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Part
WHERE NOT EXIST(SELECT * FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number=Parts.part-number))

(B) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Part
WHERE NOT EXIST(SELECT * FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number=Parts.part-number))

(C) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE NOT EXIST(SELECT *
FROM Part
WHERE EXIST(SELECT * FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number=Parts.part-number))

(D) SELECT DISTINCT suppliers.supplier-name
FROM Suppliers
WHERE EXIST(SELECT *
FROM Part
WHERE EXIST(SELECT * FROM Shipments
WHERE Shipments.supplier-number = suppliers.supplier-number
AND
Shipments.part-number=Parts.part-number))

(iii) Get part numbers for parts that either weigh more than-16 pounds or are supplied by supplier *SN*3, or both?

(A) SELECT parts.part-number
FROM parts
WHERE Parts.weight>18
UNION
SELECT Shipments.part-number
FROM shipments
WHERE Shipments.supplier-number='SN2'

(B) SELECT parts.part-number
FROM parts
WHERE Parts.weight>18
UNION
SELECT Shipments.supplier-name
FROM shipments
WHERE Shipments.supplier-number='*SN*2'

(C) SELECT parts.part-number
FROM parts
WHERE Parts.weight>18
UNION
SELECT Shipments.part-number,Shipments.supplier-name
FROM shipments
WHERE Shipments.supplier-number='*SN*2'

(D) SELECT parts.part-Number, parts.color
FROM parts
WHERE Parts.weight>18
UNION
SELECT Shipments.part-number
FROM shipments
WHERE Shipments.supplier-number='*SN*2'

Common data for questions 16 and 17: Consider the following relation: Teach

Name	Address	course
Zohar	40B,east city	MD
Nisha	16/2, hyd	BDS
Zohar	40B, East city	MS
Ravi	New York	MBA

16. The teacher with name Zohar teaching the course MS?

(A) σ_{Name} = 'Zohar' teach = MS.
(B) π_{Name} = 'Zohar' teach = MS.
(C) σ_{name} = 'Zohar' and course = 'MS' (teach).
(D) π_{Name} = 'Zohar' and course = 'MS' (teach).

17. Select the names of courses taught by Zohar?

(A) $\pi_{course}(\sigma_{Name\ =\ 'Zohar'}(Teach))$
(B) $\sigma_{course}(\pi_{Name\ =\ 'Zohar'}(Teach))$
(C) $\pi_{course}(\sigma_{Name\ =\ 'MD'}(Teach))$
(D) None

18. Consider the join of a relation *A* with a relation *B*. If *A* has *m* tuples and *B* has *n* tuples. Then the maximum and minimum sizes of the join respectively are.

(A) *mn* and $m + n$ (B) $m + n$ and $(m - n)$
(C) *mn* and *m* (D) *mn* and 0

19. Match the following:

I	Set intersection	1	$R \mid\times\mid S$
II	Natural join	2	$r - (r - s)$
III	Division	3	$\leftarrow$
IV	Assignment	4	$\pi_{R-S}(r) - \pi_{R-S}$ $(\pi_{R-S}(r) \times s)$ $-\pi_{R-S}, s(r)$

(A) I – 2, II – 1, III – 4, IV – 3
(B) I – 3, II – 4, III – 2, IV – 1
(C) I – 1, II – 2, III – 3, IV – 4
(D) I – 2, II – 3, III – 4, IV – 1

20. Which one is correct for division operations for relation r and s

(A) $r \div s$
(B) $\pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - \pi_{R-S}), s(r)$
(C) Temp 1 $\leftarrow \pi_{R-S}(r)$
Temp 2 $\leftarrow \pi_{R-S}(\text{temp1} \times s) - \pi_{R-S}, s(r)$
result = temp 1 – temp 2
(D) All the above

Practice Problems 2

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. The correct order of SQL expression is
(A) Select, group by, where, having
(B) Select, where, group by, having
(C) Select, group by, having, where
(D) Select, having, where, group by

2. Which one is not a query language?
(A) SQL (B) QBE
(C) Data log (D) MySQL

3. Like '*a b* \ % *c d*' escape '\' matches all the strings
(A) Ending with *a b c d*
(B) Beginning with *a b c d*
(C) Beginning with *a b c d*
(D) Beginning with *a b* % *c d*

4. '_ _ _%' matches any string of
(A) At least three characters
(B) At most three characters
(C) Exactly three characters
(D) exactly three characters ending with %

5. Which of the following are set operations?
(i) Union
(ii) Intersection
(iii) Set Difference
(iv) Cartesian Product
(A) (i), (ii), (iii)
(B) (i), (iii), (iv)
(C) (i), (iii), (ii), (iv)
(D) (i), (ii), (iv)

6. What is the purpose of project operation?
(A) It selects certain columns
(B) It selects certain rows
(C) It selects certain strings
(D) It selects certain integers

Common data for questions 7 and 8: Person

Id	Name	Age	Hobby
11	Anu	21	Stamp Collection
22	Kamal	32	Painting
33	Ravi	24	Dancing
44	Ram	22	Singing

7. Select the persons whose hobby is either painting (or) singing.
(A) $\sigma_{\text{Hobby = 'painting' OR Hobby = 'singing'}}$ (person)
(B) $\sigma_{\text{Hobby = 'painting','singing'}}$ (person)
(C) $\sigma_{\text{Hobby = 'painting' OR 'singing'}}$ (person)
(D) All are correct

8. Select the persons whose age is above 21 and below 32:
(A) $\sigma_{\text{age > 21 AND age < 32}}$ (person)
(B) $\sigma_{\text{21 < age < 32}}$ (person)
(C) $\sigma_{\text{age > 21 OR age < 32}}$ (person)
(D) $\sigma_{\text{age < 21 AND age > 32}}$ (person)

Common data for questions 9 and 10: Consider the following relation: Teach

Name	course	Rating	Age
Zohar	MD	7	35
Nisha	BDS	8	27
Zohar	MS	7	34
Ravi	MBA	9	33

9. Select the teachers whose rating is above 7 and whose age is less than 32?
(A) $s_{\text{Rating > 7 AND Age < 32}}$ (Teach)
(B) $s_{\text{Rating} \geq \text{7 AND Age < 32}}$ (Teach)
(C) $s_{\text{Rating > 7 AND < 32}}$ (Teach)
(D) Both (A) and (B)

10. Select the courses with rating above 7?
(A) $\pi_{\text{course}}(\sigma_{\text{rating > 7}}(\text{Teach}))$
(B) $\sigma_{\text{course}}(\pi_{\text{rating > 7}}(\text{Teach}))$
(C) $\pi_{\text{name, course}}(\sigma_{\text{rating > 7}}(\text{Teach}))$
(D) None

Common data for questions 11 and 12: Consider the following schema of a relational database employee (empno, ename, eadd) project (pno, pname) Work–on (empno, pno) Part(partno, partname, qty-on-hand, size) Use (empno, pno, partno, number)

11. Display the names of the employees who are working on a project named 'VB'.
(A) $\sigma_{\text{name}}(\text{employee} \bowtie (\sigma_{\text{pname = 'VB'}}, \text{project}) \bowtie \text{worked on})$
(B) $\sigma_{\text{name}}(\text{employee} \bowtie (\pi_{\text{pname = 'VB'}}, (\text{project}) \bowtie \text{work on})$
(C) $\pi_{\text{name}}(\text{employee} \bowtie (\sigma_{\text{pname = 'VB'}}, (\text{project}) \bowtie \text{work on})$
(D) $\pi_{\text{name}}(\text{employee} \bowtie (\pi_{\text{pname}} = \text{'VB'}(\text{project}) \bowtie \text{work on})$

12. Display the names of the people who are not working for any project.

(A) π_{name} (employee $\bowtie$(π_{name} (employee + work on)

(B) π_{name} (employee – π_{name} (employee $\cap$ work on)

(C) π_{name} (employee – π_{name} (employee $\bowtie$ work on)

(D) σ_{name} (employee – σ_{name} (employee $\bowtie$ work on)

13. Consider the following tables:

A	B	C	D
b	c	e	f
a	b	i	j
b	c	g	h
b	c	a	d
d	i	g	h
d	j	j	k
d	i	e	f

C	D
e	f
g	h

$R \div S$

A	B
b	c
d	i

Which of the following statements is true?

(A) $R \div S = p_{A,B}(R) - p_{A,B}(p_{A,B}(R) \times S + R)$

(B) $R \div S = p_{A,B}(R) - p_{A,B}(p_{A,B}R \times S - R)$

(C) $R \div S = p_{A,B}(R) - p_{A,B}((p_{A,B}(R) \times S) - R)$

(D) $R \div S = p_{A,B}(R) - p_{A,B}(p_{A,B}(R) \times R - S)$

Common data for questions 14 and 15: Consider the following schema of a relational data base
student (sno, name, address)
project (pno, Pname) work-on (sno, pno)
Part (part no, part name, qtyon hand size)
Use (sno, pno, part no, number)

14. List the names of the students who are participating in every project and have used every part.

(A) σ_{name}(student $\bowtie$(((Workon) $\div$ σ_{pro}(project)) $\cap$ (σ_{sno}, $_{part\ no}$(use) $\div$ $\sigma_{part\ no}$ (part)))

(B) π_{name}(student $\bowtie$(((Workon) $\div$ π_{pro}(project)) $\cap$ (π_{sno}, $_{partno}$(use) $\div$ $\sigma_{part\ no}$ (part)))

(C) π_{name}(student $\bowtie$(((Workon) $\div$ π_{partno} (project)) $\cap$ ($\pi_{sno,}$ $_{partno}$(use) $\div$ $\sigma_{part\ no}$ (part)))

(D) π_{name}(student ∞ (((Workon) $\div$ π_{pro}(project)) $\cup$ (π_{ssno}, $_{partno}$ (use) $\div$ $\pi_{part\ no}$ (part)))

15. The following query gives π_{name} (employee $\bowtie$(work on $\div$ π_{pro} ($\sigma_{Pname\ =\ 'MS'\ AND\ 'MD'}$(project)))

(A) Names of the students who are working in either projects 'MS' or 'MD'

(B) Names of the students who are working in both the projects 'MS' or 'MD'

(C) Names of the students who are not working in any of the projects 'MS' or 'MD'

(D) None of the above

16. 'All rows corresponding to students whose sno's are between 10 and 20

(i) Select * form student where SNo are between 5 AND 10

(ii) Select * from student where SNO IN(5, 10)

(A) Only (i) (B) Only (ii)

(C) Both (A) and (B) (D) None

17. UPDATE account SET
DA = basic * .2,
GROSS = basic * 1.3, Where basic > 2000;

(A) The above query displays *DA* and gross for all those employees whose basic is ≥ 2000

(B) The above query displays *DA* and Gross for all employees whose basic is less than 2000

(C) The above query displays updated values of *DA* as well as gross for all those employees whose basic is > 2000

(D) All the above

18. Given two union compatible relations $R_1(A, B)$ and $R_2(C, D)$, what is the result of the operation
R_1 $A = CAB = DR_2$?

(A) $R_1 \cup R_2$ (B) $R_1 \times R_2$

(C) $R_1 - R_2$ (D) $R_1 \cap R_2$

19. Which of the following queries finds the clients of banker Agassi and the city they live in?

(A) $\pi_{client'cname'Ccity}$($\sigma_{client.cname}$ = customer c name ($\sigma_{Banker.}$ $_{name = Aggassi}$ (client × customer)

(B) $\pi_{Client.c\ city}$ ($\sigma_{Banker\ name\ =\ 'Aggassi'}$(client × customer)

(C) $\pi_{client'c\ name'Cucity}$($\sigma_{client.c\ name\ =\ 'Aggasi'}$ ($\sigma_{client'name\ =\ Cutome}$r (client × customer)

(D) $\pi_{.c\ name'Cucity}$($\sigma_{Bankers}$ name = name ($\sigma_{Banker.\ =\ agassi}$ (client × customer)

20. Consider the following schema pertaining to students data
Student (<u>rno,</u> name, add)

Enroll (rno, <u>Cno,</u> Cname) Where the primary keys are shown Underlined. The no. of tuples in the student and Enroll tables are 120 and 8 respectively. What are the maximum and minimum no. of tuples that can be present in (student * Enroll) where '*' denotes natural join.

(A) 8, 8 (B) 120, 8

(C) 960, 8 (D) 960, 120

Previous Years' Questions

1. Consider the relation account (customer, balance) where customer is a primary key and there are no null values. We would like to rank customers according to decreasing balance. The customer with the largest balance gets rank 1, ties are not broke but ranks are skipped; if exactly two customers have the largest balance they each get rank 1 and rank 2 is not assigned.

Query 1: select A.customer, count (B.customer) from account *A*, account *B* where A.balance <= B.balance group by A.customer
Query 2: select A.customer, 1 + count (B.balance) from account *A*, account *B* where A.balance < B.balance group by A.customer Consider these statements about Query1 and Query2.

1. Query1 will produce the same row set as Query2 for some but not all databases.
2. Both Query1 and Query2 are correct implementation of the specification.
3. Query1 is a correct implementation of the specification but Query2 is not.
4. Neither Query1 nor Query2 is a correct implementation of the specification.
5. Assigning rank with a pure relational query takes less time than scanning in decreasing balance order assigning ranks using ODBC.

Which two of the above statements are correct? **[2006]**

(A) 2 and 5 (B) 1 and 3
(C) 1 and 4 (D) 3 and 5

2. Consider the relation enrolled (student, course) in which (student, course) is the primary key, and the relation paid (student, amount) where student is the primary key. Assume no null values and no foreign keys or integrity constraints. Given the following four queries:

Query1: select student from enrolled where student in (select student from paid)

Query2: select student from paid where student in (select student from enrolled)

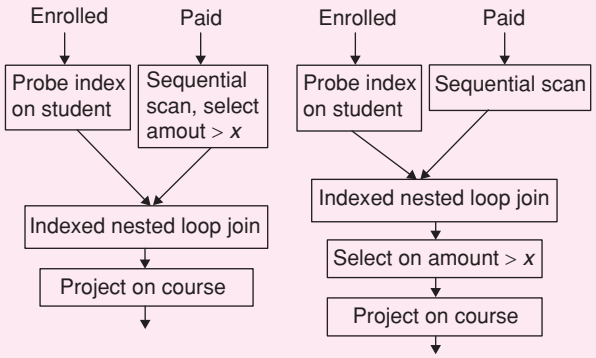
Query3: select E.student from enrolled E, paid P where E.student = P.student

Query4: select student from paid where exists (select * from enrolled where enrolled.student = paid.student)

Which one of the following statement is correct? **[2006]**

(A) All queries return identical row sets for any database
(B) Query2 and Query4 return identical row sets for all databases but there exist databases for which Query1 and Query2 return different row sets
(C) There exist databases for which Query3 returns strictly fewer rows than Query2
(D) There exist databases for which Query4 will encounter an integrity violation at runtime

3. Consider the relation enrolled (student, course), in which (student, course) is the primary key, and the relation paid (student, amount) where student is the primary key. Assume no null values and no foreign keys or integrity constraints. Assume that amounts 6000, 7000, 8000, 9000 and 10000 were each paid by 20% of the students. Consider these query plans (plan 1 on left, plan 2 on right) to 'list all courses taken by students who have paid more than *x*'



A disk seek takes 4 ms, disk data transfer bandwidth is 300 MB/s and checking a tuple to see if amount is greater than x takes 10 μs. Which of the following statements is correct? **[2006]**

(A) Plan 1 and Plan 2 will not output identical row sets for all databases
(B) A course may be listed more than once in the output of Plan 1 for some databases
(C) For $x = 5000$, Plan 1 executes faster than Plan 2 for all databases
(D) For $x = 9000$, Plan 1 executes slower than Plan 2 for all databases

4. Information about a collection of students is given by the relation **studinfo** (studId, name, sex). The relation **enroll** (studId, courseId) gives which student has enrolled for (or taken) what course(s). Assume that every course is taken by at least one male and at least one female student. What does the following relational algebra expression represent?

$$\Pi_{courseId}((\Pi_{studid}(\sigma_{sex = \text{'female'}}(studInfo)) \times \Pi_{courseId}(enroll)) - enroll)$$ **[2007]**

(A) Courses in which all the female students are enrolled
(B) Courses in which a proper subset of female students are enrolled.

(C) Courses in which only male students are enrolled.
(D) None of the above

5. Consider the relation **employee** (name, sex, supervisorName) with *name* as the key. *supervisorName* gives the name of the supervisor of the employee under consideration. What does the following Tuple Relational Calculus query produce?

$$e \cdot \text{name} \mid \text{employee}(e) \wedge$$
$$(\forall x)[\neg \text{employee}(x) \vee x \cdot \text{supervisor Name} \neq e \cdot \text{name} \vee$$
$$x \cdot \text{sex} = \text{"male"}]\}$$

[2007]

(A) Names of employees with a male supervisor.
(B) Names of employees with no immediate male subordinates.
(C) Names of employees with no immediate female subordinates.
(D) Names of employees with a female supervisor.

6. Consider the table **employee** (empId, name, department, salary) and the two queries Q_1, Q_2 below. Assuming that department 5 has more than one employee, and we want to find the employees who get higher salary than anyone in the department 5, which one of the statements is **TRUE** for any arbitrary employee table?

Q_1: SELECT e.empId
FROM employee e
WHERE not exists
(Select * From employee s where s.department = '5' and s.salary >=e.salary)

Q_2: SELECT e.empId
FROM employee *e*
WHERE e.salary > Any
(Select distinct salary From employee *s* Where s.department = '5') **[2007]**

(A) Q_1 is the correct query
(B) Q_2 is the correct query
(C) Both Q1 and Q2 produce the same answer.
(D) Neither Q_1 nor Q_2 is the correct query

7. Let R and S be two relations with the following schema
R ($\underline{P}$, $\underline{Q}$, $R1$, $R2$, $R3$)
S ($\underline{P}$, $\underline{Q}$, $S1$, $S2$)
Where $\{P, Q\}$ is the key for both schemas. Which of the following queries are equivalent?

I. $\Pi_P (R \bowtie S)$
II. $\Pi_P (R) \bowtie \Pi_P (S)$
III. $\Pi_P (\Pi_{P,Q} (R) \cap \Pi_{P,Q} (S))$
IV. $\Pi_P (\Pi_{P,Q} (R) - (\Pi_{P,Q} (R) - (\Pi_{P,Q} (S)))$ **[2008]**

(A) Only I and II (B) Only I and III
(C) Only I, II and III (D) Only I, III and IV

8. Let R and S be relational schemes such that $R = \{a,b,c\}$ and $S = \{c\}$. Now consider the following queries on the database:

I. $\pi_{R-S}(r) - \pi_{R-S}(\pi_{R-S}(r) \times s - \pi_{R-S,S}(r))$
II. $\{t \mid t \in \pi_{R-S}(r) \wedge \forall u \in s(\exists v \in r(u = v[s] \wedge t = v[R-S]))\}$
III. $\{t \mid t \in \pi_{R-S}(r) \wedge \forall v \in r(\exists u \in s(u = v[s] \wedge t = v[R-S]))\}$
IV. SELECT R.a, R.b
FROM *R*, *S*
WHERE R.c = S.c

Which of the above queries are equivalent? **[2009]**

(A) I and II (B) I and III
(C) II and IV (D) III and IV

Common data for questions 9 and 10: Consider the following relational schema: Suppliers (sid: integer, sname: string, city: string, street: string) Parts(pid: integer, pname: string, color: string) Catalog (sid: integer, pid: integer, cost: real)

9. Consider the following relational query on the above database:

SELECT S.sname
FROM Suppliers S
WHERE S.sid NOT IN (SELECT C.sid
FROM Catalog C
WHERE C.pid NOT IN (SELECT P.pid FROM Parts P
WHERE P.color <> 'blue'))

Assume that relations corresponding to the above schema are not empty. Which one of the following is the correct interpretation of the above query? **[2009]**

(A) Find the names of all suppliers who have supplied a non-blue part.
(B) Find the names of all suppliers who have not supplied a non-blue part.
(C) Find the names of all suppliers who have supplied only blue parts.
(D) Find the names of all suppliers who have not supplied only blue parts.

10. A relational schema for a train reservation database is given below

Passenger (pid, pname, age)
Reservation (pid, cass, tid)

Table :Passenger
Table :Reservation

Pid	pname	Age
0	'Sachin'	65
1	'Rahul'	66
2	'Sourav'	67

Pid	class	tid
0	'AC'	8200
1	'AC'	8201
2	'SC'	8201

3	'Anil'	69	5	'AC'	8203
			1	'SC'	8204
			3	'AC'	8202

What pids are returned by the following SQL query for the above instance of the tables?

```
SELECT  pid
FROM    Reservation
WHERE   class = 'AC' AND
EXISTS (SELECT *
        FROM Passenger
        WHERE age > 65 AND
        Passenger.pid = Reservation.pid)
```

[2010]

(A) 1, 0 (B) 1, 2
(C) 1, 3 (D) 1, 5

11. Consider a relational table r with sufficient number of records, having attributes $A_1, A_2, \ldots A_n$ and let $1 \le p \le n$. Two queries $Q1$ and $Q2$ are given below.

$Q1$: $\pi_{A_1 \ldots A_n}(\sigma_{A_p=c}(r))$ where c is a constant.

$Q2$: $\pi_{A_1 \ldots A_n}(\sigma_{c_1 \le A_p \le c_2}(r))$ where c_1 and c_2 are constants.

The database can be configured to do ordered indexing on A_P or hashing on A_p. Which of the following statements is TRUE? **[2011]**

(A) Ordered indexing will always outperform hashing for both queries
(B) Hashing will always outperform ordered indexing for both queries.
(C) Hashing will outperform ordered indexing on $Q1$, but not on $Q2$.
(D) Hashing will outperform ordered indexing on $Q2$, but not on $Q1$.

12. Database table by name Loan_Records is given below.

Borrower	Bank manager	Loan amount
Ramesh	Sunderajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sunderajan	7000.00

What is the output of the following SQL query?

```
SELECT count ( * )
FROM (Select Borrower, Bank_Manager FROM
Loan Records) AS S
NATURAL JOIN
(SELECT Bank_Manager, Loan_Amount FROM
Loan_Records) AS T;
```

[2011]

(A) 3 (B) 9
(C) 5 (D) 6

13. Consider a database table T containing two columns X and Y each of type *integer*. After the creation of the table, one record ($X = 1$, $Y = 1$) is inserted in the table. Let MX and MY denote the respective maximum values of X and Y among all records in the table at any point in time. Using MX and MY, new records are inserted in the table 128 times with X and Y values being $MX + 1$, $2 * MY + 1$ respectively. It may be noted that each time after the insertion, values of MX and MY change. What will be the output of the following SQL query after the steps mentioned above are carried out?

SELECT Y FROM T WHERE $X = 7$; **[2011]**

(A) 127 (B) 255
(C) 129 (D) 257

14. Which of the following statements are true about an SQL query?

P: An SQL query can contain a HAVING clause even if it does not have a GROUP BY clause
Q: An SQL query can contain a HAVING clause only if it has a GROUP BY clause
R: All attributes used in the GROUP BY clause must appear in the SELECT clause
S: Not all attributes used in the GROUP BY clause need to appear in the SELECT clause **[2012]**

(A) P and R (B) P and S
(C) Q and R (D) Q and S

15. Suppose $R_1(\underline{A}, B)$ and $R_2(\underline{C}, D)$ are two relation schemas. Let r_1 and r_2 be the corresponding relation instances. B is a foreign key that refers to C in R_2. If data in r_1 and r_2 satisfy referential integrity constraints, which of the following is always true? **[2012]**

(A) $\Pi_B(r_1) - \Pi_C(r_2) = \varnothing$
(B) $\Pi_C(r_2) - \Pi_B(r_1) = \varnothing$
(C) $\Pi_B(r_1) = \Pi_C(r_2)$
(D) $\Pi_B(r_1) - \Pi_C(r_2) \ne \varnothing$

Common data for questions 16 and 17: Consider the following relations A, B and C:

(A)

Id	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

(B)

Id	Name	Age
15	Shreya	24
25	Hari	40
98	Rohit	20
99	Rohit	11

(C)

Id	Phone	Area
10	2200	02
99	2100	01

16. How many tuples does the result of the following SQL query contain?

SELECT A.Id
FROM A
WHERE A. Age > ALL (SELECT B. Age
FROM B
WHERE B. Name = 'Arun') **[2012]**

(A) 4 (B) 3
(C) 0 (D) 1

17. How many tuples does the result of the following relational algebra expression contain? Assume that the schema of $A \cup B$ is the same as that of A.

$(A \cup B) \bowtie_{A.Id > 40 \vee C.Id < 15} C$ **[2012]**

(A) 7 (B) 4
(C) 5 (D) 9

18. Consider the following relational schema. Students (rollno: integer, sname: string) Courses (courseno: integer, cname: string) Registration(rollno:integer,co urseno: integer, percent: real)

Which of the following queries are equivalent to this query in English?

'Find the distinct names of all students who score more than 90% in the course numbered 107'

(I) SELECT DISTINCT S.sname FROM Students as *S*, Registration as *R* WHERE R.rollno=S.rollno AND R.courseno=107 AND R.percent>90

(II) $\pi_{sname}(\sigma_{courseno=107 \wedge percent>90}$ Registration $\bowtie$ Students)

(III) $\{T \mid \exists S \in$ Students, $\exists R \in$ Registration (S.rollno=*R*. rollno $\wedge$ R.courseno=107 $\wedge$ R.percent>90$\wedge$T. sname=S.sname)$\}$

(IV) $\{<S_N> \mid \exists S_R \exists R_P (<S_R, S_N> \in$ Students $\wedge <S_R, 107, R_P> \in$ Registration $\wedge R_P > 90)\}$ **[2013]**

(A) I, II, III and IV (B) I, II and III only
(C) I, II and IV only (D) II, III and IV only

19. Given the following statements:

S_1: A foreign key declaration can always be replaced by an equivalent check assertion in SQL.

S_2: Given the table *R* (*a*, *b*, *c*) where a and b together form the primary key, the following is a valid table definition.

CREATE TABLE *S* (
a INTEGER
d INTEGER,
e INTEGER,
PRIMARY KEY (d),
FOREIGN KEY (a) references *R*)

Which one of the following statements is CORRECT? **[2014]**

(A) S_1 is TRUE and S_2 is FALSE
(B) Both *S*1 and S_2 are TRUE
(C) S_1 is FALSE and S_2 is TRUE
(D) Both S_1 and S_2 are FALSE

20. Given the following schema:

Employees (emp–id, first-name, last– name, hire–date, dept–id, salary)

Departments (dept–id, dept–name, manager–id, location–id)

you want to display the last names and hire dates of all latest hires in their respective departments in the location ID 1700. You issue the following query:

```
SQL > SELECT last–name, hire–date
FROM employees
WHERE (dept–id, hire–date) IN
(SELECT dept–id, MAX (hire–date)
FROM employees JOIN departments USING
(dept–id)
WHERE location–id = 1700
GROUP BY dept–id);
```

What is the outcome? **[2014]**

(A) It executes but does not give the correct result.
(B) It executes and gives the correct result.
(C) It generates an error because of pair wise comparison.
(D) It generates an error because the GROUP BY clause cannot be used with table joins in a subquery.

21. Given an instance of the STUDENTS relation as shown below:

Student ID	Student Name	Student Email	Student Age	CPI
2345	Shankar	shaker @ math	X	9.4
1287	Swati	swati @ ee	19	9.5
7853	Shankar	shankar @ cse	19	9.4
9876	Swati	swati @ mech	18	9.3
8765	Ganesh	ganesh@ civil	19	8.7

For (StudentName, StudentAge) to be a key for this instance, the value X should NOT be equal to _____. **[2014]**

22. Consider a join (relation algebra) between relations ($r(R)$) and ($s(S)$) using the nested loop method. There are three buffers each of size equal to disk block size, out of which one buffer is reserved for intermediate results. Assuming size $r(R) < size\ s(S)$, the join will have fewer number of disk block accesses if **[2014]**

(A) Relation $r(R)$ is in the outer loop
(B) Relation $s(S)$ is in the outer loop
(C) Join selection factor between $r(R)$ and $s(S)$ is more than 0.5
(D) Join selection factor between $r(R)$ and $s(S)$ is less than 0.5

23. SQL allows duplicate tuples in relations, and correspondingly defines the multiplicity of tuples in the result of joins. Which one of the following queries always gives the same answer as the nested query shown below:

Select * from R where a in (select S. a from S) **[2014]**

(A) Select $R.*$ from R, S where $R.a = S.a$

(B) Select distinct R * from R, S where $R.a = S.a$

(C) Select $R.*$ from R, (select distinct a from S) as $S1$ where $R.a = S1.a$

(D) Select $R.*$ from R, S where $R.a = S.a$ and is unique R

24. What is the optimized version of the relation algebra expression $\pi_{A_1}(\pi_{A_2}(\sigma_{F_1}(\sigma_{F_2(r)})))$, where A_1, A_2 are sets of attributes in r with $A_1 \subset A_2$ and F_1, F_2 are Boolean expressions based on the attributes in r? **[2014]**

(A) $\pi_{A_1}(\sigma_{(F_1 \wedge F_2)^{(r)}})$

(B) $\pi_{A_1}(\sigma_{(F_1 \vee F_2)^{(r)}})$

(C) $\pi_{A_2}(\sigma_{(F_1 \wedge F_2)^{(r)}})$

(D) $\pi_{A_2}(\sigma_{(F_1 \vee F_2)^{(r)}})$

25. Consider the relational schema given below, where **eId** of the relation **dependent** is a foreign key referring to **empId** of the relation **employee**. Assume that every employee has at least one associated dependent in the **dependent** relation.

Consider the following relational algebra query:

employee (empId, empName, empAge)

dependent (depId, eId, depName, depAge)

π_{empId}(employee)- π_{empId} (employee $\bowtie_{(empId = eID) \wedge (empAge \leq depAge)}$ dependent)

The above query evaluates to the set of empIds of employees whose age is greater than that of **[2014]**

(A) some dependent.

(B) all dependents.

(C) some of his/her dependents.

(D) all of his/her dependents.

26. Consider the following relational schema:

employee (empId, empName, empDept)

customer(custId, custName, salesRepid, rating)

salesRepId is a foreign key referring to empId of the employee relation. Assume that each employee makes a sale to at least one customer. What does the following query return?

```
SELECT empName
FROM employee E
WHERE NOT EXISTS
(SELECT custId
FROM customer C
WHERE C.salesRepId = E.empId
AND C.Rating < > 'GOOD');
```

[2014]

(A) Names of all the employees with at least one of their customers having a 'GOOD' rating.

(B) Names of all the employees with at most one of their customers having a 'GOOD' rating.

(C) Names of all the employees with none of their customers having a 'GOOD' rating.

(D) Names of all the employees with all their customers having a 'GOOD' rating.

27. SELECT operation in SQL is equivalent to **[2015]**

(A) The selection operation in relational algebra

(B) The selection operation in relational algebra, except that SELECT in SQL retains duplicates.

(C) The projection operation in relational algebra.

(D) The projection operation in relational algebra, except that SELECT in SQL retains duplicates.

28. Consider the following relations:

Student

Roll No	Student Name
1	Raj
2	Rohit
3	Raj

Performance

Roll No	Course	Marks
1	Math	80
1	English	70
2	Math	75
3	English	80
2	Physics	65
3	Math	80

Consider the following SQL query.

```
SELECT S.Student_Name, sum (P.Marks)
FROM Student S, Performance P
WHERE S.Roll_No = P.Roll_No
GROUP BY S.Student_Name
```

The number of rows that will be returned by the SQL query is _____ **[2015]**

29. Consider two relations $R_1(A, B)$ with the tuples (1, 5), (3, 7) and $R_2(A, C) = (1, 7), (4, 9)$. Assume that $R(A, B, C)$ is the full natural outer join of R_1 and R_2. Consider the following tuples of the form (A, B, C): a = (1, 5, null), b = (1, null, 7), c = (3, null, 9), d = (4, 7, null), e = (1, 5, 7), f = (3, 7, null), g = (4, null, 9). Which one of the following statements is correct? **[2015]**

(A) R contains a, b, e, f, g but not c, d.

(B) R contains all of a, b, c, d, e, f, g.

(C) R contains e, f, g but not a, b.

(D) R contains e but not f, g.

30. Consider the following relation

Cinema (theater, address, capacity)

Which of the following options will be needed at the end of the SQL query

SELECT P_1.address
FROM Cinema P_1
such that it always finds the addresses of theaters with maximum capacity? **[2015]**

(A) WHERE P_1.capacity > = All (select P_2. Capacity from Cinema P_2)
(B) WHERE P_1.capacity >= Any (select P_2. Capacity from Cinema P_2)
(C) WHERE P_1.capacity > All (select max(P_2. capacity) from Cinema P_2)
(D) WHERE P_1.capacity > Any (select max(P_2. capacity) from Cinema P_2)

31. Which of the following is NOT a superkey in a relational schema with attributes *V, W, X, Y, Z* and primary key *VY*? **[2016]**

(A) *V XYZ* (B) *V WXZ*
(C) *V WXY* (D) *V WXYZ*

32. Consider a database that has the relation schema EMP (EmpId, EmpName and DeptName). An instance of the schema EMP and a SQL query on it are given below.

EMP		
EmpId	**EmpName**	**DeptName**
1	XYA	AA
2	XYB	AA
3	XYC	AA
4	XYD	AA
5	XYE	AB
6	XYF	AB
7	XYG	AB
8	XYH	AC
9	XYI	AC
10	XYJ	AC
11	XYK	AD
12	XYL	AD
13	XYM	AE

```
SELECTIVE AVG(EC.Num)
FROM EC
WHERE (DeptName, Num) IN
      (SELECT DeptName, COUNT (EmpId) AS
                              EC(DeptName, Num)
       FROM EMP
       GROUP BY DeptName)
```

The output of executing the SQL query is _____. **[2017]**

33. Consider a database that has the relation schemas EMP(EmpId, EmpName, DeptId), and DEPT(DeptName, DeptId), Note that the DeptId can be permitted to be NULL in the relation EMP. Consider the following queries on the database expressed in tuple relational calculus.

(I) $\{t \mid \exists u \in \text{EMP}(t[\text{EmpName}] = u[\text{EmpName}] \wedge \forall v \in \text{DEPT}(t[\text{DeptId}] \neq v[\text{DeptId}]))\}$
(II) $\{t \mid \exists u \in \text{EMP}(t[\text{EmpName}] = u[\text{EmpName}] \wedge \exists v \in \text{DEPT}(t[\text{DeptId}] \neq v[\text{DeptId}]))\}$
(III) $\{t \mid \exists u \in \text{EMP}(t[\text{EmpName}] = u[\text{EmpName}] \wedge \exists v \in \text{DEPT}(t[\text{DeptId}] = v[\text{DeptId}]))\}$

Which of the above queries are safe? **[2017]**

(A) (I) and (II) only
(B) (I) and (III) only
(C) (II) and (III) only
(D) (I), (II) and (III)

34. Consider a database that has the relation schema CR (studentName, CourseName). An instance of the schema CR is as given below.

CR	
StudentName	**CourseName**
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD
SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

The following query is made on the database.

$T1 \leftarrow \pi_{CouraseName}(\sigma_{StudentName='SA'}(CR))$

$T2 \leftarrow CR \div T1$

The number of rows in $T2$ is _____. **[2017]**

35. Consider the following database table named *top_scorer*.

top_scorer

player	country	goals
Klose	Germany	16
Ronaldo	Brazil	15
G Miiller	Germany	14
Fontaine	France	13
Pelé	Brazil	12
Klinsmann	Germany	11
Kocsis	Hungary	11
Batistuta	Argentina	10
Cubillas	Peru	10
Lato	Poland	10
Lineker	England	10
T Muller	Germany	10
Rahn	Germany	10

Consider the following SQL query:

```
SELECT ta.player FROM top_scorer AS ta
WHERE ta.goals >ALL (SELECT tb.goals
    FROM top_scorer AS tb
    WHERE tb.country = 'Spain')
AND ta.goals >ANY (SELECT tc.goals
    FROM top_scorer AS tc
    WHERE tc. country = 'Germany')
```

The number of tuples returned by the above SQL query is _____. **[2017]**

36. Consider the following two tables and four queries in SQL.

Book (isbn, bname), Stock (isbn, copies)

Query 1:
```
SELECT B.isbn, S.copies
FROM Book B INNER JOIN Stock S
ON B.isbn = S.isbn;
```

Query 2:
```
SELECT B.isbn, S.copies
FROM Book B LEFT OUTER
JOIN Stock S
ON B.isbn = S.isbn;
```

Query 3:
```
SELECT B.isbn, S.copies
FROM Book B RIGHT OUTER
JOIN Stock S
ON B.isbn = S.isbn;
```

Query 4:
```
SELECT B.isbn, S.copies
FROM Book B FULL OUTER
JOIN Stock S
ON B.isbn = S.isbn;
```

Which one of the queries above is certain to have an output that is a superset of the outputs of the other three queries? **[2018]**

(A) Query 1 (B) Query 2
(C) Query 3 (D) Query 4

37. Consider the relations $r(A, B)$ and $s(B, C)$, where $s \cdot B$ is a primary key and $r \cdot B$ is a foreign key referencing $s \cdot B$. Consider the query

$$Q: r \bowtie (\sigma_{B<5}(S))$$

Let LOJ denote the natural left outer-join operation. Assume that r and s contain no null values.

Which one of the following queries is NOT equivalent to Q? **[2018]**

(A) $\sigma_{B<5}(r \bowtie s)$ (B) $\sigma_{B<5}(r \text{ LOJ } s)$
(C) $r \text{ LOJ } (\sigma_{B<5}(s))$ (D) $\sigma_{B<5}(r) \text{ LOJ } s$

Answer Keys

Exercises

Practice Problems 1

1. A **2.** B **3.** C **4.** (i) B (ii) D (iii) D **5.** C **6.** C **7.** (i) A (ii) C
8. (i) B (ii) A **9.** A **10.** A **11.** (i) A (ii) B **12.** (i) B (ii) B (iii) A
13. (i) A (ii) A (iii) A **14.** (i) A (ii) A (iii) A **15.** (i) A (ii) A (iii) A **16.** C
17. A **18.** D **19.** A **20.** D

Practice Problems 2

1. B **2.** D **3.** D **4.** A **5.** C **6.** A **7.** A **8.** A **9.** A **10.** A
11. C **12.** C **13.** C **14.** C **15.** C **16.** B **17.** C **18.** D **19.** B **20.** A

Previous Years' Questions

1. C **2.** A **3.** C **4.** B **5.** C **6.** B **7.** D **8.** A **9.** A **10.** C
11. C **12.** C **13.** A **14.** C **15.** A **16.** B **17.** A **18.** A **19.** D **20.** B
21. 19 **22.** A **23.** C **24.** A **25.** D **26.** D **27.** D **28.** 2 **29.** C **30.** A
31. B **32.** 2.6 **33.** D **34.** 4 **35.** 7 **36.** D **37.** C

Chapter 3

Normalization

LEARNING OBJECTIVES

- *Normalization*
- *Anomalies*
- *First normal form*
- *Functional dependency*
- *Inference rules*
- *Second normal form*
- *Third normal form*
- *Higher normal forms (Boyce-Codd normal form)*
- *Fifth normal form*
- *Courses*

Normalization

Database design theory includes design standards called *normal forms*. The process of making data and tables match these standards is called *normalizing data* or *data normalization*. By normalizing data, we eliminate redundant information and organize table to make it easier to manage the data and make future changes to the table and database structure. This process removes the insertion, deletion, and modification anomalies. In normalizing your data, we usually divide large tables into smaller, easier to maintain tables. We can then use the technique of adding foreign keys to enable connections between the tables.

Data normalization is part of the database design process and is neither specific nor unique to any particular RDBMS. These are in order, such as first, second, third, Boyce-Codd, fourth, and fifth normal forms. Each normal form represents an increasingly stringent set of rules; that is, each normal form assumes that the requirements of the preceding forms have been met. Many relational database designers feel that, if their tables are in third normal form, most common design problems have been addressed. However, the higher-level normal forms can be of use and are included here.

Database normalization is the process of removing redundant data from tables to improve storage efficiency, data integrity and scalability.

1. In the relational model, methods exists for quantifying how efficient a database is, these classifications are called q'.
2. Normalization generally involves splitting existing tables into multiple ones, which must be rejoined (or) linked each time a query is issued.
3. Edgar F. Codd originally established three normal forms: 1NF, 2NF, 3NF. There are others also, but 3NF is widely considered to be sufficient for most applications, most tables when reaching 3NF are also in BCNF (Boyce–Codd normal form).

Table 1

Title	Author 1	Author 2	I SBN	Subject	Pages	Publisher
Database system concepts	Abraham Silber schatz	Henry F. Korth	0072958863	My SQL, computers	1160	McGraw-Hill
OS concepts	Abraham Silberschatz	Henry F. Korth	0471694665	Computers	990	McGraw-Hill

Problems:

1. This table is not very efficient with storage.
2. This design doesn't protect data integrity.
3. This table doesn't scale well.

Anomalies

An anomaly is a variation that differs in some way from what is said to be normal, with respect to maintaining a database.

1. The basic operations performed on Databases are Record insertion, Record updation, Record deletion.

2. It is desirable for these operations to be straight forward and efficient.
3. When relations are not fully normalized they exhibit anomalies.
4. The design goal of database is too easily to understand and to maintain.
5. Anomalies are problems that occur in un-normalized databases where all the data is stored in one table.

Types of anomalies

There are three types of anomalies that can arise in the database because of redundancy as follows:

1. Insertion anomaly
2. Deletion anomaly
3. Updation anomaly

Insertion anomaly An insertion anomaly occurs when particular attributes cannot be inserted into the database without the presence of other attributes.

Example: Consider the following table: Sales

Sales-Rep-Id	Name	Hire-Date	Client
1	Ana	1/1/2015	Madison
2	Sudha	2/4/2014	Peterson
3	Joey	3/2/2014	John
* New			

Insertion anomaly occurs in the above table which stores records for a company's sales representatives and the clients for whom they are responsible.

1. It is not possible to add records for newly hired Sales representatives until they have been assigned to one or more clients.
2. If we insert a record for newly hired, client column will be NULL, which is a required field for the table.
3. It is not possible to record newly hired in the table during training.

Deletion anomaly Deletion anomaly occurs when some particular attributes are lost because of the deletion of other attributes.

Example: Consider the following table 'course'.

S No	C No	S Name	Course
S41	C9201	John	Sales
S42	C9401	Brat	Finance
S40	C9201	Amit	Sales
S43	C9608	Arun	Accounts

Execute the following *SQL* query:

```
Delete    *
From      course
Where     S No = S43
```

If we delete a tuple where SNo = S43, he is the only (or) last student in the accounts department, we will lose data about student 'S43, Arun' as well as data about Accounts course that is 'C9608, Accounts'.

Updation anomaly An updation anomaly occurs when one or more instances of duplicated data are updated but not all.

Example: Consider the 'course' table given in the above example.

If we want to update course – No (Cno) of sales C9201 to C8686, in the course table.

1. It might happen that, the tuple with S No = S41 updated its CNo to C8686, but not the tuple with SNo = S43.
2. Inconsistency occurs in the table, because for the same course sales we have 2 different course Numbers.

Determining keys For a table 'R', its schema R consists of all attributes of R, we say X is a key to R if $X \rightarrow R$ means

X determines R
R is dependent upon X
If you know x then you know R

Example: Consider a relation schema R(ABCDE) and the functional dependencies:

$AC \rightarrow D$

$B \rightarrow E$

$DA \rightarrow B$

The closure of AC determines all the attributes present in Relation R, so the key for R is 'AC'.

$$
\begin{aligned}
AC^{+} = \quad & \{AC\} && \text{(self determination)} \\
& \{ACD\} && (AC \rightarrow D) \\
& \{ACDB\} && (DA \rightarrow B) \\
& \{ACDBE\} && (B \rightarrow E)
\end{aligned}
$$

$\therefore$ key = AC

Any attribute which does not appear on the right-hand-side of a given functional dependency appears in any one of the candidate keys.

1. From the above example, neither A (or) C appears in the right hand side of any functional dependency.

First Normal Form (1NF)

In Table 1, we have two violations of 1NF such as:

1. More than one author field and
2. Subject field contains more than one piece of information with more than one value in a single field; thus, it would be very difficult to search for all books on a given subject.

Table 2 *1NF table*

Title	Author	ISBN	Subject	Pages	Publisher
Database system concept	Abrahem Silbers Chatt	0072958863	My SQL	1160	McGraw-Hill
Database system concept	Henry K. Forth	0072958863	Computers	1160	McGraw-Hill
OS concepts	Henry K. Forth	0471694665	Computers	990	McGraw-Hill
OD concepts	Abraham Silber Schatz	0471694665	Computers	990	McGraw-Hill

In Table 2, we have two rows for a single book. Additionally, we would be violating the second NF. A better solution to the problem would be to separate the data into separate tables—an author table and a subject table to store our information, removing that information from the book table.

Table 3 *Subject table*

Subject–ID	Subject
1	My SQL
2	computers

Table 4 *Author table*

Author–ID	Last Name	First name
1	Silberschatz	Abraham
2	Korth	Henry

Table 5 *Book table*

ISBN	Title	Pages	Publisher
00729 58863	Database System Concepts	1160	McGraw-Hill
04716 94665	OS concepts	990	McGraw-Hill

Each table has a primary key, used for joining tables together when querying the data.

A table is in first normal form (1NF) if there are no repeating groups. A repeating group is a set of logically related fields or values that occur multiple times in one record. The sample tables below do not comply with first normal form. Look for fields that contain too much data and repeating group of fields.

EMPLOYEES_PROJECTS_TIME

A table with fields containing too much data.

Employee ID	Name	Project	Time
EN1-26	Sean O'Brien	30-452-T3, 30-457-T3, 32-244-T3	0.25, 0.40, 0.30
EN1-33	Amy Guya	30-452-T3, 30-382-TC, 32-244-T3	0.05, 0.35, 0.60
EN1-35	Steven Baranco	30-452-T3, 31-238-TC	0.15, 0.80
EN1-36	Elizabeth Roslyn	35-152-TC	0.90
EN1-38	Carol Schaaf	36-272-TC	0.75
EN1-40	Alexandra Wing	31-238-TC, 31-241-TC	0.20, 0.70

The example above is also related to another design issue, namely, that each field should hold the smallest meaningful value and that there should not be multiple values in a single field.

Why is this table design a problem?

There would be no way to sort by last names or to know which allocation of time belonged to which project.

EMPLOYEES_PROJECTS_TIME

Table 5 *A table with repeating groups of fields.*

Emp ID	Last Name	First Name	Project1	Time1	Project2	Time2	Project3	Time3
EN1-26	O'Brien	Sean	30-452-T3	0.25	30-457-T3	0.40	32-244-T3	0.30
EN1-33	Guya	Amy	30-452-T3	0.05	30-382-TC	0.35	32-244-T3	0.60
EN1-35	Baranco	Steven	30-452-T3	0.15	31-238-TC	0.80		
EN1-36	Roslyn	Elizabeth	35-152-TC	0.90				
EN1-38	Schaaf	Carol	36-272-TC	0.75				
EN1-40	Wing	Alexandra	31-238-TC	0.20	31-241-TC	0.70		

If an employee was assigned to a fourth project, you would have to add two new fields to the table. Also, it would be very difficult to total the amount of time devoted to a particular project.

The design problems addressed are very common, particularly among new designers who are accustomed to tracking data in a spreadsheet. Often, when building a spreadsheet, we arrange the data horizontally, laying it out across the spreadsheet. When designing tables, we have to think more vertically. Similar data belongs in the same column or field with a single value in each row.

Now we will take the table you saw above and redesign it so it will comply with first normal form.

Look at the repeating groups of data. Identify tables and fields that will hold this data without the repeating groups. Think vertically and remember that similar data belongs in the same field.

Enter the sample data from the table to make sure you don't have repeating groups. If necessary, include foreign key field(s) to connect the tables.

EMPLOYEES

EmployeeID	Last Name	First Name
EN1-26	O'Brien	Sean
EN1-33	Guya	Amy
EN1-35	Baranco	Steven
EN1-36	Roslyn	Elizabeth
EN1-38	Schaaf	Carol
EN1-40	Wing	Alexandra

PROJECTS_EMPLOYEES_TIME

Project Num	EmployeeID	Time
30-328-TC	EN1-33	0.35
30-452-T3	EN1-26	0.25
30-452-T3	EN1-33	0.05
30-452-T3	EN1-35	0.15
31-238-TC	EN1-35	0.80
30-457-T3	EN1-26	0.40
31-238-TC	EN1-40	0.20
31-241-TC	EN1-40	0.70
32-244-T3	EN1-33	0.60
35-152-TC	EN1-36	0.90
36-272-TC	EN1-38	0.75

Mark the primary key field(s) and foreign keys in each table. Shown below with * indicating the Primary key.

EMPLOYEES

EmployeeID	Last Name	First Name
EN1-26	O'Brien	Sean
EN1-33	Guya	Amy
EN1-35	Baranco	Steven
EN1-36	Roslyn	Elizabeth
EN1-38	Schaaf	Carol
EN1-40	Wing	Alexandra

PROJECTS_EMPLOYEES_TIME

Project Num	EmployeeID	Time
30-328-TC	EN1-33	0.35
30-452-T3	EN1-26	0.25
30-452-T3	EN1-33	0.05
30-452-T3	EN1-35	0.15
31-238-TC	EN1-35	0.80
30-457-T3	EN1-26	0.40
31-238-TC	EN1-40	0.20
31-241-TC	EN1-40	0.70
32-244-T3	EN1-33	0.60
35-152-TC	EN1-36	0.90
36-272-TC	EN1-38	0.75

If an employee was assigned to an additional project, it would involve merely adding a new record. Also, it would be much easier to search for a particular project number as they are all held in a single column.

Functional Dependency

A functional dependency is a relationship between fields so that the value in Field *A* determines the value in Field *B*, and there can be only one value in Field *B*. In that case, Field *B* is functionally dependent on Field *A*. Consider the following sample table:

Airport	City
National	Washington, DC
JFK	New York
LaGuardia	New York
Logan	Boston
Dulles	Washington, DC

Each airport name is unique and each airport can be in only one city. Therefore, City is functionally dependent on Airport. The value in the Airport field determines what the value will be in the City field (making Airport the determinant field) and there can be only one value in the City field. This does not need to work in the reverse. As shown in the table, a city can have more than one airport, so Airport is not functionally

dependent on City; the value in City does not necessarily determine what the value in Airport will be.

You will sometimes see a functional dependency written in this format:

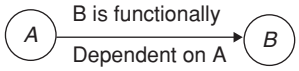
Determinant field(s) → Functionally dependent field

as in:

Airport → City

Functional dependency describes the relationship between attributes in a relation.

Example: If A and B are attributes of relation R, and B is functionally dependent on A ($A \rightarrow B$) if each value of A is associated with one value of B.



Determinant refers to the attributed (or) group attributes on the left-hand side of the arrow of a functional dependency.

Inference Rules

The following inference rules IR 1 through IR 6 form a complete set for inferring functional and multi-valued dependencies from a given set of dependencies

Assume that all attributes are included in a 'universal' relation schema $R = \{A_1, A_2, \ldots A_N\}$ and that X, Y, Z and W are subsets of R.

IR 1 (reflexive rule): if $X \supseteq Y$, then $X \rightarrow Y$
IR 2 (Augmentation rule): $\{X \rightarrow Y\} = XZ \rightarrow YZ$
IR 3 (transitive rule): $\{X \rightarrow Y, Y \rightarrow Z\} = X \rightarrow Z$
IR 4 (complementation rule): $\{X \rightarrow \rightarrow Y\} = \{X \rightarrow \rightarrow (R - (X \cup Y))\}$
IR5 (augmentation rule for MVD's): if $X \rightarrow \rightarrow y$ and $W \rightarrow Z$ then $WX \supseteq YZ$
IR6 (transitive rule for MVD's):
$\{X \rightarrow \rightarrow Y, Y \rightarrow \rightarrow Z\} = X \rightarrow \rightarrow (Z - Y)$

Second Normal Form

A table is said to be in second normal form if it is in first normal form and each non-key field is functionally dependent on the entire primary key.

Look for values that occur multiple times in a non-key field. This tells us that we have too many fields in a single table.

Example: In the example below, see all the repeating values in the name and Project Title fields. This is an inefficient way to store and maintain data. In a well-designed database, the only data that is duplicated is in key fields used to connect tables. The presumption is that the data in key fields will rarely change, while the data in non-key fields may change frequently.

A table with a multifield primary key and repeating data in non-key fields

EmployeeID	Last Name	First Name	Project Number	Project Title
EN1-26	O'Brien	Sean	30-452-T3	STAR manual
EN1-26	O'Brien	Sean	30-457-T3	ISO procedures
EN1-26	O'Brien	Sean	31-124-T3	Employee handbook
EN1-33	Guya	Amy	30-452-T3	STAR manual
EN1-33	Guya	Amy	30-482-TC	Web Site
EN1-33	Guya	Amy	31-241-TC	New catalogue
EN1-35	Baranco	Steven	30-452-T3	STAR manual
EN1-35	Baranco	Steven	31-238-TC	STAR prototype
EN1-36	Roslyn	Elizabeth	35-152-TC	STAR pricing
EN1-38	Schaaf	Carol	36-272-TC	Order system
EN1-40	Wing	Alexandra	31-238-TC	STAR prototype
EN1-40	Wing	Alexandra	31-241-TC	New catalogue

If a ProjectTitle changed, we would have to edit it in several records. And what would happen in this table if the EmployeeID was part of the primary key and we wanted to add a new ProjectNum and ProjectTitle even though no employees had yet been assigned?

The primary key cannot contain a null value so you couldn't add the new project. Additionally, if a project ended and you wanted to delete it, you would have to delete the individual values because, if we deleted the records containing the titles and an employee was assigned to only that project, you would also delete that employee's record, something that we may not want to do.

In the above example, the asterisks indicate the fields that make up the primary key of this table as it now stands. A multifield primary key is necessary because neither the EmployeeID nor the ProjectNum fields contain unique values.

The reason there are repeated values in LastName, FirstName, and ProjectTitle is that these fields are dependent

on only part of the primary key. The value in EmployeeID determines what the value in LastName will be, but the value in ProjectNum has nothing to do with it. Similarly, the value in ProjectNum determines the value in ProjectTitle, but EmployeeID does not. These non-key fields relate to only part of the primary key. They are not functionally dependent on the entire primary key.

The solution to this lies in breaking the table into smaller tables that do meet second normal form. You will find that more tables are the solution to most problems encountered during data normalisation.

EMPLOYEES

EmployeeID	Last Name	First Name
EN1-26	O'Brien	Sean
EN1-33	Guya	Amy
EN1-35	Baranco	Steven
EN1-36	Roslyn	Elizabeth
EN1-38	Schaaf	Carol
EN1-40	Wing	Alexandra

EMPLOYEES_PROJECTS

EmployeeID	Project Num
EN1-26	30-452-T3
EN1-26	30-457-T3
EN1-26	31-124-T3
EN1-33	30-328-TC
EN1-33	30-452-T3
EN1-33	32-244-T3
EN1-35	30-452-T3
EN1-35	31-238-TC
EN1-36	35-152-TC
EN1-38	36-272-TC
EN1-40	31-238-TC
EN1-40	31-241-TC

Now we'll take the table above and design new tables that will eliminate the repeated data in the non-key fields.

1. To decide what fields belong together in a table, think about which field determines the values in other fields. Create a table for those fields and enter the sample data.
2. Think about what the primary key for each table would be and about the relationship between the tables. If necessary, add foreign keys or a junction table.
3. Mark the primary key for each table and make sure that you don't have repeating data in non-key fields.

PROJECTS

Projec tNum	Project Title
30-452-T3	STAR manual
30-457-T3	ISO procedures
30-482-TC	Web site
31-124-T3	Employee handbook
31-238-TC	STAR prototype
31-238-TC	New catalog
35-152-TC	STAR pricing
36-272-TC	Order system

Examine the tables to make sure there are no repeating values in non-key fields and that the value in each non-key field is determined by the value(s) in the key field(s). This removes the modification anomaly of having the repeated values.

Third Normal Form

A table is said to be in third normal form if it is in second normal form (2NF) and there are no transitive dependencies.

A transitive dependency is a type of functional dependency in which the value in a non-key field is determined by the value in another non-key field and that field is not a candidate key. Again, look for repeated values in a non-key field as in the following example.

A table with a single field primary key and repeating values in non-key fields.

Project Num	Project Title	Project Mgr	Phone
30-452-T3	STAR manual	Garrison	2756
30-457-T3	ISO procedures	Jacanda	2954
30-482-TC	Web site	Friedman	2846
31-124-T3	Employee handbook	Jones	3102
31-238-TC	STAR prototype	Garrison	2756
31-241-TC	New catalog	Jones	3102
35-152-TC	STAR pricing	Vance	3022
36-272-TC	Order system	Jacanda	2954

The phone number is repeated each time a manager's name is repeated. It is dependent on the manager, which is dependent on the project number (a transitive dependency).

The Project Manager field is not a candidate key, because the same person manages more than one project. Again, the solution is to remove the field with repeating data to a separate table.

Take the above table and create new tables to fix the problem.

1. Think about which fields belong together and create new tables to hold them.
2. Enter the sample data and check for unnecessarily (not part of primary key) repeated values.
3. Identify the primary key for each table and, if necessary, add foreign keys.

PROJECTS

Project Num	Project Title	Project Mgr
30-452-T3	STAR manual	Garrison
30-457-T3	ISO procedures	Jacanda
30-482-TC	Web site	Friedman
31-124-T3	Employee handbook	Jones
31-238-TC	STAR prototype	Garrison
31-241-TC	New catalog	Jones
35-152-TC	STAR pricing	Vance
36-272-TC	Order system	Jacanda

MANAGERS

Project Manager	Phone
Friedman	2846
Garrison	2756
Jacanda	2954
Jones	3102
Vance	3022

Reexamine your tables to make sure there are no unnecessarily repeating values in non-key fields and that the value in each non-key field is determined by the value(s) in the key field(s). In most cases, 3NF should be sufficient to ensure that your database is properly normalised.

Higher Normal Forms (Boyce–Codd Normal Form)

A table is in third normal form (3NF), and all determinants are candidate keys.

Boyce–Codd normal form (BCNF) can be thought of as a 'new' third normal form. It was introduced to cover situations that the 'old' third normal form did not address. The mean of a determinant (determines the value in another field) and candidate keys (qualify for designation as primary key). This normal form applies to situations where you have overlapping candidate keys.

If a table has no non-key fields, it is automatically in BCNF (Figure 1). Look for potential problems in updating existing data (modification anomaly) and in entering new data (insertion anomaly).

Imagine that we were designing a table for a college to hold information about courses, students, and teaching assistants. We have the following business rules:

1. Each course can have many students.
2. Each student can take many courses.
3. Each course can have multiple teaching assistants (TAs).
4. Each TA is associated with only one course.
5. For each course, each student has one TA.

Some sample data:

COURSES_STUDENTS_TA's

CourseNum	Student	TA
ENG101	Jones	Clark
ENG101	Grayson	Chen
ENG101	Samara	Chen
MAT350	Grayson	Powers
MAT350	Jones	O'Shea
MAT350	Berg	Powers

To uniquely identify each record, we could choose CourseNum + Student as a primary key. This would satisfy third normal form also because the combination of CourseNum and Student determines the value in TA. Another candidate key would be Student + TA. In this case, you have overlapping candidate keys (Student is in both). The second choice, however, would not comply with third normal form, because the CourseNum is not determined by the combination of Student and TA; it only depends on the value in TA. This is the situation that Boyce-Codd normal form addresses; the combination of Student + TA could not be considered to be a candidate key.

If we wanted to assign a TA to a course before any students enrolled, we couldn't because Student is part of the primary key. Also, if the name of a TA changed, would have to update it in multiple records. If assume have just these fields, this data would be better stored in three tables: one with CourseNum and Student, another with Student and TA, and third with CourseNum and TA.

COURSES

Course Num	Student
ENG101	Jones
ENG101	Grayson
ENG101	Samara
MAT350	Grayson
MAT350	Jones
MAT350	Berg

STUDENTS

Student	TA
Jones	Clark
Grayson	Chen
Samara	Chen
Grayson	Powers
Jones	O'Shea
Berg	Powers

TA's

*CourseNum	*TA
ENG101	Clark
ENG101	Chen
MAT350	O'Shea
MAT350	Powers

Figure 1 Tables that comply with BCNF.

Fourth Normal Form

A table is in Boyce-Codd normal form (BCNF) and there are no multi-valued dependencies.

A *multi-valued dependency* occurs when, for each value in field *A*, there is a set of values for field *B* and a set of values for field *C* but fields *B* and *C* are not related.

Look for repeated or null values in non-key fields. A multi-valued dependency occurs when the table contains fields that are not logically related. An often used example is the following table:

MOVIES

Movie	Star	Producer
Once Upon a Time	Julie Garland	Alfred Brown
Once Upon a Time	Mickey Rooney	Alfred Brown
Once Upon a Time	Julie Garland	Muriel Humphreys
Once Upon a Time	Mickey Rooney	Muriel Humphreys
Moonlight	Humphrey Bogart	Alfred Brown
Moonlight	Julie Garland	Alfred Brown

A movie can have more than one star and more than one producer. A star can be in more than one movie. A producer can produce more than one movie. The primary key would have to include all three fields, and so this table would be in BCNF. But you have unnecessarily repeated values, with the data maintenance problems that causes and you would have trouble with deletion anomalies.

The Star and the Producer really aren't logically related. The Movie determines the Star and the Movie determines the Producer. The answer is to have a separate table for each of those logical relationships: one holding Movie and Star and the other with Movie and Producer, as shown below:

STARS

*Movie	*Star
Once Upon a Time	Julie Garland
Once Upon a Time	Mickey Rooney
Moonlight	Humphrey Bogart
Moonlight	Julie Garland

PRODUCERS

*Movie	*Producer
Once Upon a Time	Alfred Brown
Once Upon a Time	Muriel Humphreys
Moonlight	Alfred Brown

Above, showing tables that comply with 4NF

Below is another example of a common design error, and it's easily spotted by all the missing or blank values.

PROJECTS_EQUIPMENT

Dept Code	Project Num	Project Mgr ID	Equipment	Property ID
IS	36-272-TC	EN1-15	CD-ROM	657
IS			VGA desktop monitor	305
AC	35-152-TC	EN1-15		
AC			Dot-matrix printer	358
AC			Calculator with tape	239
TW	30-452-T3	EN1-10	486 PC	275
TW	30-457-T3	EN1-15		
TW	31-124-T3	EN1-15	Laser printer	109
TW	31-238-TC	EN1-15	Handheld scanner	479
RI			Fax machine	775
MK			Laser printer	858
MK			Answering machine	187
TW	31-241-TC	EN1-15	Standard 19200 bps modem	386
SL			486 Laptop PC	772
SL			Electronic notebook	458

A table with many null values (Note: It also does not comply with 3NF and BCNF).

It is the same problem here because not all of the data is logically related. As usual, the answer is more tables: one to hold the information on the equipment assigned to departments (with PropertyID as the primary key) and another with projects and departments. We would now the business rules to know whether a project might involve more than one department or manager and be able to figure out the primary key. Assuming a project can have only one manager and be associated with only one department, the tables would be as follows:

EQUIPMENT

*Property ID	Equipment	DeptCode
657	CD-ROM	IS
305	VGA desktop monitor	IS
358	Dot-matrix printer	AC
239	Calculator with tape	AC
275	486 PC	TW
109	Laser printer	TW
479	Handheld scanner	TW
775	Fax machine	RI
858	Laser printer	MK
187	Answering machine	MK
386	Standard 19200 bps modem	TW
772	486 Laptop PC	SL
458	Electronic notebook	SL

PROJECTS_EQUIPMENT

Project Num	Project Mgr ID	Dept Code
36-272-TC	EN1-15	IS
35-152-TC	EN1-15	AC
30-452-T3	EN1-10	TW
30-457-T3	EN1-15	TW
31-124-T3	EN1-15	TW
31-238-TC	EN1-15	TW
31-241-TC	EN1-15	TW

Figure 2 Tables that eliminate the null values and comply with 4NF.

Fifth Normal Form

A table is in fourth normal form (4 NF) and there are no cyclic dependencies.

A *cyclic dependency* can occur only when you have a multifield primary key consisting of three or more fields. For example, let's say your primary key consists of fields *A*, *B*, and *C*. A cyclic dependency would arise if the values in those fields were related in pairs of *A* and *B*, *B* and *C*, and *A* and *C*.

Fifth normal form is also called *projection-join normal form*. A *projection* is a new table holding a subset of fields from an original table. When properly formed projections are joined, they must result in the same set of data that was contained in the original table.

Look for the number of records that will have to be added or maintained

Following is some sample data about buyers, the products they buy, and the companies they buy from.

BUYING

Buyer	Product	Company
Chris	Jeans	Levi
Chris	Jeans	Wrangler
Chris	Shirts	Levi
Lori	Jeans	Levi

Figure 3 A table with cyclic dependencies.

The primary key consists of all three fields. One data maintenance problem that occurs is that you need to add a record for every buyer who buys a product for every company that makes that product or they can't buy from them. That may not appear to be a big deal in this sample of two buyers, two products, and two companies ($2 \times 2 \times 2 = 8$ total records). But what if we went to 20 buyers, 50 products, and 100 companies ($20 \times 50 \times 100 = 100{,}000$ potential records)? It quickly gets out of hand and becomes impossible to maintain.

We might solve this by dividing this into the following two tables:

BUYERS

Buyer	Product
Chris	jeans
Chris	shirts
Lori	jeans

PRODUCTS

Product	Company
jeans	Wrangler
jeans	Levi
shirts	Levi

However, if you joined the two tables above on the Product field, it would produce a record not part of the original data set (it would say that Lori buys jeans from Wrangler). This is where the projection-join concept comes in.

The correct solution would be three tables:

BUYERS

*Buyer	*Product
Chris	jeans
Chris	shirts
Lori	jeans

PRODUCTS

*Product	*Company
jeans	Wrangler
jeans	Levi
shirts	Levi

COMPANIES

*Buyer	*Company
Chris	Levi
Chris	Wrangler
Lori	Levi

Figure 4 Tables that comply with 5NF.

When the first two tables are joined by Product and the result joined to the third table by Buyer and Company, the result is the original set of data.

EXERCISES

Practice Problems I

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Consider the given functional dependencies

$A \rightarrow B$

$BC \rightarrow DE$

$AEF \rightarrow G$

Which of the following is true?

(A) Functional dependency $ACF \rightarrow DG$ implied by the set
(B) Functional dependency $ACF \rightarrow DG$ cannot be implied by the set
(C) Functional dependency $AB \rightarrow G$ implied by the set
(D) Both (*B*) and (*C*)

2. Consider the given relation

DNAME	DNO	MGRNO	LOCATION
RESEARCH	5	333	{BANGLORE,DELHI, HYDERABAD}
ADMINISTRATION	4	987	{CHENNAI}
EXECUTIVES	1	885	{HYDERABAD}

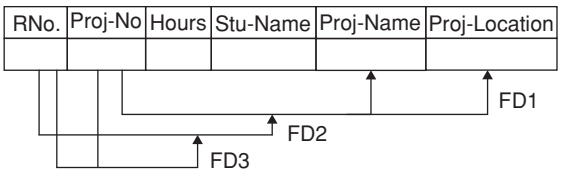
Department

The given relation is

(A) is not in 1NF (B) in 1NF
(C) in 2NF (D) in 3NF

3. Consider the given Relational scheme

Student-project

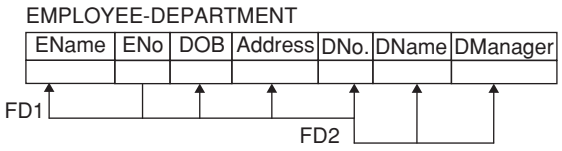


Which functional dependencies are violating 2NF property?

(A) FD1 (B) FD2
(C) FD3 (D) Both A and B

4. Consider the given relation

EMPLOYEE-DEPARTMENT



Which functional dependencies are violating 3NF ?

(A) FD1 (B) FD2
(C) Both (D) None of these

5. Consider the given relation $R(A, B, C, D)$and functional dependencies:

$FD = (AB \rightarrow C$

$C \rightarrow B$

$C \rightarrow D)$

Determine the key, prime attributes and non-prime attributes.

(A) $\{A\}, \{AB\}, \{CDE\}$
(B) $\{AB, AC\}, \{ABC\}, \{D\}$
(C) $\{AB, BC\}, \{ABC\}, \{D\}$
(D) $\{AB, AC\}, \{AB\}, \{D\}$

6. Consider the given relation and functional dependencies

$R(ABCDE)$

$FD = (ABD \rightarrow C$

$BC \rightarrow D$

$CD \rightarrow E)$

Determine the key, prime attributes, non-prime attributes and the normal form of the relation?

(A) $\{AB, AD\}, \{ABCD\}, \{E\}$
(B) $\{ABC, ABD\}, \{ABCD\}, \{E\}$
(C) $\{AB, AD\}, \{ABC\}, \{DE\}$
(D) $\{ABC, ABD\}, \{AB\}, \{CDE\}$

7. Consider the given relation and functional dependencies
$R(ABC)$
$FD = (AB \rightarrow C$
$C \rightarrow A)$
The relation is in which normal form?
(A) 1NF (B) 2NF
(C) 3NF (D) BCNF

8. Consider the given relation and its functional dependencies:
$R(ABCDE)$
$FD = (AB \rightarrow C$
$C \rightarrow E$
$B \rightarrow D$
$E \rightarrow A)$
The relation is further decomposed into two relations:
$R_1(BCD), R_2(ACE)$
(A) Decomposition is lossy and dependency preserving
(B) Decomposition is lossless and dependency preserving
(C) Decomposition is lossy and not dependency preserving
(D) Decomposition is lossless and not dependency preserving

9. Consider the following relational instance:

X	Y	Z
1	4	2
1	5	3
1	6	3
3	2	2

Which of the following functional dependencies are satisfied by the instance?
(A) $xy \rightarrow z$ and $z \rightarrow y$
(B) $yz \rightarrow x$ and $y \rightarrow z$
(C) $yz \rightarrow x$ and $x \rightarrow z$
(D) $xz \rightarrow y$ and $y \rightarrow x$

10. Consider the following functional dependencies:
DOB → Age
Age → Eligibility
Name → RNo
RNo → Name
CourseNo → CourseName
CourseNo → Instructor
(RNo, CNo) → Grade
The relation (RNo, Name, DOB, Age) is in which normal form?
(A) 1NF (B) 2NF
(C) 3NF (D) BCNF

11. Consider the given functional dependencies:
$AB \rightarrow CD$
$AF \rightarrow D$
$DE \rightarrow F$
$C \rightarrow G$
$F \rightarrow E$
$G \rightarrow A$
Which of the following is false?
(A) $\{CF\}^+ = \{ACDEFG\}$
(B) $\{BG\}^+ = \{ABCDG\}$
(C) $\{AF\}^+ = \{ACDEFG\}$
(D) $\{AB\}^+ = \{ABCDG\}$

12. What should be the key to make the given relation to be in BCNF? The dependencies for the following, 'Grades' relation are GRADES (student-Id, course#, semester#, Grade) student-Id, course#, semester# → Grade
(A) student-Id
(B) course#
(C) semester#
(D) student-Id, course#, semester #

13. What normal from is the following relation in?
STORE_ITEM (SKU, promotionID, vendor, style, price)
SKU, promotionID → vendor, style, price
SKU → vendor, style
(A) 1NF (B) 2NF
(C) 3NF (D) 4NF

14. What normal form is the following relation in?
Only H, I can act as the key
STUFF (H, I, J, K, L, M, N, O)
$H, I \rightarrow$ J,K,L
$J \rightarrow M$
$K \rightarrow N$
$L \rightarrow O$?
(A) 1NF (B) 2NF
(C) 3NF (D) BCNF

15. What normal form the following relation is in?
STUFF2(D, O, N, T, C, R, Y)
$D, O \rightarrow N, T, C, R, Y$
$C, R \rightarrow D$
$D \rightarrow N$?
(A) 1NF (B) 2NF
(C) 3NF (D) BCNF

16. The given table is in the BCNF form, convert it to the 4th normal form.

Employee	Skill	Language
Jones	Electrical	French
Jones	Electrical	German
Jones	Mechanical	French
Jones	Mechanical	German
Smith	Plumbing	Spanish

(A)

Employee	Skill

(B)

Employee	Language

(C)

Skill	Language

(D) Both A and B

17. For a database relation $x(a, b, c, d)$, where all the domains of a, b, c, d, include only atomic values, only the following FDs and those that can be inferred from them hold.

$a \rightarrow b, c \rightarrow d$

the relation is

(A) In 1st NF but not in 2nd NF
(B) In 2nd NF but not in 3rd NF
(C) In 2nd NF
(D) In 3rd NF

18. Which of the following FDs are satisfied by the instance from the below relation:

A	B	C
2	8	4
2	10	6
2	12	6
6	4	4

(A) AB → C and C → B
(B) BC → A and B → C
(C) BC → A and A → C
(D) AC → B and B → A

19. Consider the following database:

Course # → Title
Course # time → location
Emp – ID → T –Name salary
is in

(A) 3NF (B) 2NF
(C) 1NF (D) BCNF

20. Consider the following schema

$A = (w, x, y, z)$ and the dependencies are

$W \rightarrow X, X \rightarrow Y, Y \rightarrow Z$, and $Z \rightarrow W$

Let $A = (A_1$ and $A_2)$ be a decomposition such that $A_1 \cap A_2 = \phi$

The decomposition is

(A) In 1NF and in 2NF
(B) In 2NF and not in 3NF
(C) In 2NF and in 3NF
(D) Not in 2NF and in 3NF

Practice Problems 2

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Integrity constraints ensures that changes made to the database by authorized users do not result in

(A) Loss of FDs
(B) Loss of keys
(C) Loss of tables
(D) Loss of data consistency

2. Relation $R = (\overline{A, B}, C, D)$ with AB as primary key. Choose one FD such that R should be in 1NF but not in 2NF

(A) $AB \rightarrow C$
(B) $AB \rightarrow D$
(C) $A \rightarrow D$
(D) $AB \rightarrow CD$

3. A normalized relation (1NF) can be retrieved from unnormalized relation by removing

(A) repeating groups (B) duplicate tuples
(C) transitive dependency (D) primary key

4. A relation will be in 2NF, if we

(A) remove repeating groups
(B) remove partial dependency
(C) remove transitive dependency
(D) have overlapping candidate key

5. Relation $R = (A, B, C, D)$ with AB as primary key, choose the FD so that R should be in 2NF but not in 3NF.

(A) $D \rightarrow C$ (B) $AB \rightarrow C$
(C) $AB \rightarrow D$ (D) $A \rightarrow B$

6. If a relation is in 2NF, then it can be in 3NF by removing

(A) repeating groups
(B) partial dependencies
(C) transitive dependencies
(D) overlapping dependencies

7. BCNF can be achieved from 3NF by removing

(A) repeating groups
(B) partial dependencies
(C) transitive dependencies
(D) overlapping dependencies

8. Which one of the following is not possible?
(A) Relation is in BCNF but not in 4NF
(B) Relation is in 3NF but not in BCNF
(C) Relation is in 2NF but not in 3NF
(D) Relation is in 3NF but not in 2NF

Common data for questions 9 and 10: Let R be a relation schema R (A, B, C, D);

$F = \{AB \rightarrow CD;\ C \rightarrow A\}$ F is the set of functional dependencies

9. How many prime attributes are there?
(A) 1 (B) 2
(C) 3 (D) 4

10. The highest normal form of the above relation is
(A) 1NF (B) 2NF
(C) 3NF (D) 4NF

Linked answer questions

11. For a given relation schema $R = \{A, B, C, D, E\}$
$A \rightarrow BC$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$
Which of the following is not a candidate key?
(A) A (B) B
(C) E (D) BC

12. For the above answer, what is the closure?
(A) BD (B) ABC
(C) $ABCDE$ (D) BC

13. Consider the following functional dependencies:
$A \rightarrow B$
$C \rightarrow D$
$B \rightarrow E$
$F \rightarrow A$

The relation (A, B, C, D) is
(A) in second normal form, but not in third normal form
(B) in third normal form, but not in $B\ C\ N\ F$
(C) in $B\ C\ N\ F$
(D) None of the above

Common data for questions 14 and 15:

$R = (A, B, C, D, E, F)$
FDs = $A \rightarrow B$
$C \rightarrow$ DF
$AC \rightarrow E$
$D \rightarrow F$

14. Determine the key from the given FDs:
(A) AB (B) AC
(C) ACB (D) ACD

15. Decompose the FDs into 2NF
(A) $R_1(AB)\ R_2(CDF)\ R_3(ACE)$
(B) $R_1(AB)\ R_2(CDEF)$
(C) $R_1(ABC)\ R_2(CDF)$
(D) $R_1(AB)\ R_2(CD)\ R_3(EF)$

16. For a database relation $x(a, b, c, d)$, where all the domains of a, b, c, d, include only atomic values, only the following FDs and those that can be inferred from them hold.

$a \rightarrow b$, $c \rightarrow d$

The relation is decomposed into $R_1(ab)$, $R_2(cd)$. Which of the following is true, The decomposition
(A) is dependency preserving
(B) is not dependency preserving
(C) is loss less
(D) Both A and C

17. Which of the following FDs are satisfied by the instance from the below relation?

A	B	C
4	12	8
4	14	10
4	16	10
10	8	8

(A) $AB \rightarrow C$ and $C \rightarrow B$
(B) $BC \rightarrow A$ and $B \rightarrow C$
(C) $BC \rightarrow A$ and $A \rightarrow C$
(D) $AC \rightarrow B$ and $B \rightarrow A$

18. Indicate which of the following statements are false: 'A relational database, which is in 3NF still have undesirable data redundancy because there may exist.
(A) Below all
(B) Non trivial FDs involving prime attributes on the right side.
(C) Non-trivial FDs involving prime attributes on the left side
(D) Non-trivial FDs involving only prime attributes

19. Consider the following database:
SOFTWARE (software-vendor, product, Release-date, systemReq, warranty)
FD: (software-vendor, product, Releasedate) → system Req, price, Warranty.
Which of the following are non–prime attributes?
(A) SystemReq
(B) Price
(C) Warranty
(D) All the above

20. Consider a relation schema R (A, B, C, D, E, X, Y) with the following FDs

$F = \{0 \rightarrow A, XD \rightarrow C, DA \rightarrow B, A \rightarrow X, XE \rightarrow B, E \rightarrow A, B \rightarrow D, DA \rightarrow B, EB \rightarrow C, AB \rightarrow C, Y \rightarrow B, C \rightarrow B\}$ is in
(A) 2NF
(B) 3NF
(C) 4NF
(D) BCNF

Previous Years' Questions

1. Which one of the following statements is false? **[2007]**

(A) Any relation with two attributes is in BCNF
(B) A relation in which every key has only one attribute is in 2NF
(C) A prime attribute can be transitively dependent on a key in a 3NF relation.
(D) A prime attribute can be transitively dependent on a key in a BCNF relation.

2. Consider the following relational schemas for a library database:

Book (Title, Author, Catalog_ no, Publisher, Year, Price)

Collection (Title, Author, Catalog_ no)

with the following functional dependencies:

I. Title Author → Catalog_no
II. Catalog_no → Title Author Publisher Year
III. Publisher Title Year → Price

Assume {Author, Title} is the key for both schemas. Which of the following statements is true? **[2008]**

(A) Both Book and Collection are in BCNF
(B) Both Book and Collection are in 3NF only
(C) Book is in 2NF and Collection is in 3NF
(D) Both Book and Collection are in 2NF only

3. The following functional dependencies hold for relations $R(A, B, C)$ and $S(B, D, E)$

$B \rightarrow A$,
$A \rightarrow C$

The relation R contains 200 tuples and the relation S contains 100 tuples. What is the maximum number of tuples possible in the natural join $R \bowtie S$? **[2010]**

(A) 100 (B) 200
(C) 300 (D) 2000

4. Which of the following is true? **[2012]**

(A) Every relation in 3NF is also in BCNF
(B) A relation R is in 3NF if every non-prime attribute of R is fully functionally dependent on every key of R
(C) Every relation in BCNF is also in 3NF
(D) No relation can be in both BCNF and 3NF

Common data questions 5 and 6: Relation R has eight attributes ABCDEFGH, Fields of R contain only atomic values.

$F = \{CH \rightarrow G, A \rightarrow BC, B \rightarrow CFH, E \rightarrow A, F \rightarrow EG\}$ is a set of functional dependencies (FDs) so that F^+ is exactly the set of FDs that hold for R.

5. How many candidate keys does the relation R have? **[2013]**

(A) 3 (B) 4
(C) 5 (D) 6

6. The relation R is **[2013]**

(A) in 1NF, but not in 2NF
(B) in 2NF, but not in 3NF
(C) in 3NF, but not in BCNF
(D) in BCNF

7. Assume that in the suppliers relation above, each supplier and each street within a city has a unique name, and (sname, city) forms a candidate key. No other functional dependencies are implied other than those implied by primary and candidate keys. Which one of the following is true about the above schema? **[2009]**

(A) The schema is in BCNF
(B) The schema is in 3NF but not in BCNF
(C) The schema is in 2NF but not in 3NF
(D) The schema is not in 2NF

8. Consider the relation scheme $R = (E, F, G, H, I, J, K, L, M, N)$ and the set of functional dependencies $\{\{E, F\} \rightarrow \{G\}, \{F\} \rightarrow \{I, J\}, \{E, H\} \rightarrow \{K, L\}, \{K\} \rightarrow \{M\}, \{L\} \rightarrow \{N\}\}$ on R. What is the key for R? **[2014]**

(A) $\{E, F\}$ (B) $\{E, F, H\}$
(C) $\{E, F, H, K, L\}$ (D) $\{E\}$

9. Given the following two statements:

S_1: Every table with two single-valued attributes is in 1NF, 2NF, 3NF and BCNF

S_2: $AB \rightarrow C, D \rightarrow E, E \rightarrow C$ is a minimal cover for the set of functional dependencies $AB \rightarrow C, D \rightarrow E, AB \rightarrow E, E \rightarrow C$

Which one of the following is correct? **[2014]**

(A) S_1 is true and S_2 is false
(B) Both S_1 and S_2 are true
(C) S_1 is false and S_2 is true
(D) Both S_1 and S_2 are false

10. The maximum number of super-keys for the relation schema R (E, F, G, H) with E as the key is _____. **[2014]**

11. A *prime attribute* of a relation scheme R is an attribute that appears **[2014]**

(A) in all candidate keys of R
(B) in some candidate key of R
(C) in a foreign key of R
(D) only in the primary key of R

12. Consider an entity-Relationship (ER) model in which entity sets E_1 and E_2 are connected by an m:n relationship R_{12}. E_1 and E_3 are connected by a 1:n (1 on the side of E_1 and n on the side of E_3) relationship R_{13}.

E_1 has two single-valued attributes a_{11} and a_{12} of which a_{11} is the key attribute. E_2 has two single-valued

attributes a_{21} and a_{22} of which a_{21} is the key attribute. E_3 has two single-valued attributes a_{31} and a_{32} of which a_{31} is the key attribute. The relationships do not have any attributes.

If a relational model is derived from the above ER model, then the minimum number of relations that would be generated if all the relations are in 3 NF is _____. **[2015]**

13. Consider the relation $X(P, Q, R, S, T, U)$ with the following set of functional dependencies

```
F = {
    {P, R} → {S, T}
    {P, S, U} → {Q, R}
    }
```

Which of the following is the trivial functional dependency in $F+$, where $F+$ is closure of F? **[2015]**

(A) $\{P, R\} \rightarrow \{S, T\}$
(B) $\{P, R\} \rightarrow \{R, T\}$
(C) $\{P, S\} \rightarrow \{S\}$
(D) $\{P, S, U\} \rightarrow \{Q\}$

14. A database of research articles in a journal uses the following schema. **[2016]**

(VOLUME, NUMBER, STARTPAGE, ENDPAGE, TITLE, YEAR, PRICE)

The primary key is (VOLUME, NUMBER, STARTPAGE,ENDPAGE) and the following functional dependencies exist in the schema.

(VOLUME, NUMBER, STARTPAGE, ENDPAGE) → TITLE

(VOLUME, NUMBER) → YEAR

(VOLUME, NUMBER,

STARTPAGE,ENDPAGE) → PRICE

The database is redesigned to use the following schemas.

(VOLUME, NUMBER, STARTPAGE, ENDPAGE, TITLE, PRICE)

(VOLUME, NUMBER, YEAR)

Which is the weakest normal form that the new database satisfies, but the old one does not?

(A) 1NF (B) 2NF
(C) 3NF (D) BCNF

15. Consider the following database table water_ schemes: **[2016]**

water_schemes

scheme_no	District name	Capacity
1	Ajmer	20
1	Bikaner	10
2	Bikaner	10
3	Bikaner	20
1	Churu	20
2	Churu	20
1	Dungargarh	10

The number of tuples returned by the following SQL query is _____ .

with total (name, capacity) **as**
select district _ name, **sum** (capacity)
from water _ schemes
group by district _ name
with total _avg (capacity) **as**
select avg (capacity)
from total
select name
from total, total _ avg
where total . capacity ≥ total_avg. capacity

16. The following functional dependencies hold true for the relational schema $R\{V, W, X, Y, Z\}$:

$$V \rightarrow W$$
$$VW \rightarrow X$$
$$Y \rightarrow VX$$
$$Y \rightarrow Z$$

Which of the following is irreducible equivalent for this set of set of functional dependencies? **[2017]**

(A) $V \rightarrow W$, $V \rightarrow X$, $Y \rightarrow V$, $Y \rightarrow Z$
(B) $V \rightarrow W$, $W \rightarrow X$, $Y \rightarrow V$, $Y \rightarrow Z$
(C) $V \rightarrow W$, $V \rightarrow X$, $Y \rightarrow V$, $Y \rightarrow X$, $Y \rightarrow Z$
(D) $V \rightarrow W$, $W \rightarrow X$, $Y \rightarrow V$, $Y \rightarrow X$, $Y \rightarrow Z$

17. Consider the following tables T1 and T2.

T1

P	Q
2	2
3	8
7	3
5	8
6	9
8	5
9	8

T2

R	S
2	2
8	3
3	2
9	7
5	7
7	2

In table T1, **P** is the primary key and **Q** is the foreign key referencing **R** in table T2 with on-delete cascade and on-update cascade. In table T2, **R** is the primary key and **S** is the foreign key referencing **P** in table Tl

with on-delete set NULL and on-update cascade. In order to delete record ⟨3, 8⟩ from table T1, the number of additional records that need to be deleted from table T1 is _____. **[2017]**

18. Consider the following four relational schemas. For each schema, all non-trivial functional dependencies are listed. The underlined attributes are the respective primary keys.

Schema I:

Registration (rollno, courses)

Field 'courses' is a set-valued attribute containing the set of courses a student has registered for.

Non-trivial functional dependency:

Rollno → courses

Schema II:

Registration (rollno, courseid, email)

Non-trivial functional dependencies:

Rollno, courseid → email

email → rollno

Schema III:

Registration (rollno, courseid, marks, grade)

Non-trivial functional dependencies:

Rollno, courseid → marks, grade

Marks → grade

Schema IV:

Registration (rollno, courseid, credit)

Non-trivial functional dependencies:

Rollno, courseid → credit

Courseid → credit

Which one of the relational schemas above is in 3NF but not in BCNF? **[2018]**

(A) Schema I
(B) Schema II
(C) Schema III
(D) Schema IV

Answer Keys

Exercises

Practice Problems 1

1. A	**2.** A	**3.** D	**4.** B	**5.** B	**6.** B	**7.** C	**8.** D	**9.** B	**10.** A
11. C	**12.** D	**13.** A	**14.** B	**15.** A	**16.** D	**17.** A	**18.** B	**19.** C	**20.** C

Practice Problems 2

1. D	**2.** C	**3.** A	**4.** B	**5.** A	**6.** C	**7.** D	**8.** D	**9.** C	**10.** C
11. B	**12.** A	**13.** D	**14.** B	**15.** A	**16.** A	**17.** C	**18.** C	**19.** D	**20.** B

Previous Years' Questions

1. D	**2.** C	**3.** A	**4.** C	**5.** B	**6.** A	**7.** B	**8.** B	**9.** A	**10.** 8
11. B	**12.** 4	**13.** C	**14.** B	**15.** 2	**16.** A	**17.** 0	**18.** B		

Chapter 4

Transaction and Concurrency

LEARNING OBJECTIVES

- *Transactions and concurrency control*
- *Transaction*
- *Transaction properties*
- *Uncommitted data*
- *Transaction processing systems*
- *Concurrency control with locking methods*
- *Two-phase locking to ensure serializability*
- *Concurrency control with time stamping methods*
- *Concurrency control with optimistic methods*
- *Recoverability*
- *Equivalence of schedules*
- *Testing for conflict serializability*

Introduction

A transaction is a logical unit of work. It begins, with the execution of a BEGIN TRANSACTION operation, and ends with the execution of a COMMIT or ROLLBACK operation. The logical unit of work that is, a transaction does not necessarily involve just a single database operation. Rather, it involves a sequence of several such operations as follows:

1. Database updates are kept in buffers in main memory and not physically written to disk until the transaction commits. That way, if the transaction terminates unsuccessfully, there will be no need to undo any disk updates.
2. Database updates are physically written to disk as part of the process of honouring the transaction's COMMIT request. That way if the system subsequently crashes, we can be sure that there will be no need to redo any disk updates.

Transactions and Concurrency Control

Database transactions reflect real-world transactions that are triggered by events, such as buying a product, registering for a course, or making a deposit in your checking account. Transactions are likely to contain many parts, for example, a sales transaction consists of at least two parts.

UPDATE inventory by subtracting number of units sold from the PRODUCT table's available quantity on hand and UPDATE the ACCOUNTS RECEIVABLE table in order to bill the CUSTOMER. All parts of a transaction must be completed to prevent data integrity problems. Therefore, executing and managing transactions are important database system activities.

Concurrency control is the management of concurrent transactions execution. When many users are able to access the database, the number of concurrent transactions tends to grow rapidly; as a result, concurrency control is especially important in multiuser database environments.

Transaction

A transaction is a logical unit of work that must be either entirely completed or aborted, no intermediate states are acceptable, that is, multicomponent transactions like the previously mentioned sale, must not be partially completed. If you read from and/or write to (update) the database, you create a transaction. Another example is using SELECT, to generate a list of table contents. Many real-world database transactions are formed by two or more database requests. A database request is the equivalent of a single SQL statement in an application program or transaction. Each database request generates several input/output operations. A transaction that changes the contents of a database must alter the database from one consistent state to another. A consistent database state is one in which all data integrity constraints are satisfied.

Example:

1. Checking an account balance:
   SELECT ACC_NUM, ACC_BALANCE
   FROM CHECKACC
   WHERE ACC_NUM = '0908110638';
   Even though we did not make any changes to the CHECKACC table, the SQL code represents a transaction, because we accessed the database.

2. Registering a credit sale of 100 units of product *X* to customer *Y* in the amount of $500.00 first, product X's quantity on hand (QOH) needs to be reduced by 100.

   UPDATE PRODUCT
   SET PROD_QOH = PROD_QOH_100
   WHERE PROD_CODE = '*x*';
   Then, $500 needs to be added to customer Y's accounts receivable
   UPDATE ACCT_RECEIVABLE
   SET ACCT_RECEIVABLE = ACCT_BALANCE + 500
   WHERE ACCT_NUM = '*Y*';

In Example 2, both the SQL, transactions must be completed in order to represent the real-world sales transaction. If both transactions are not completely executed, the transaction yields an inconsistent database.

If a transaction yields an inconsistent database, the DBMS must be able to recover the database to a previous consistent state.

Transaction Properties

All transactions must display atomicity, consistency, isolation and durability. These are known as ACID properties of transactions.

Atomicity

It requires that all operations of a transaction be completed; if not, the transaction is aborted. Therefore, a transaction is treated as a single, logical unit of work.

Consistency

It describes the result of the concurrent execution of several transactions. The concurrent transactions are treated as though they were executed in serial order. This property is important in multiuser and distributed database, where several transactions are likely to be executed concurrently.

Isolation

It means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed. Therefore, if a transaction T_1 is being executed and is using the data item X_1, that data item cannot be accessed by any other transaction (T_2… T_n) until T_1 ends. This property is particularly useful in multiuser database environment, because several different users can access and update the database at the same time.

Durability

It indicates the permanence of the database's consistent state. When a transaction is completed, the database reaches a consistent state, and that state cannot be lost, even in the event of the system's failure.

Transaction Management with SQL

The ISO standard defines a transaction model based on two SQL statements: COMMIT and ROLLBACK. The standard specifies that an SQL transaction automatically begins with a transaction-initiating SQL statement executed by a user or program (e.g., SELECT, INSERT, UPDATE). Changes made by a transaction are not visible to other concurrently executing transactions until the transaction completes. When a transaction sequence is initiated, it must continue through all succeeding SQL, statements until one of the following four events occur:

1. A COMMIT statement ends the transaction successfully, making the database changes permanent.
   A new transaction starts after COMMIT with the next transaction initiating statement.
2. For programmatic SQL, successful program termination ends the final transaction successfully, even if a commit statement has not been executed (equivalent to COMMIT)
3. For programmatic SQL, abnormal program termination aborts the transaction (equivalent to ROLLBACK)

Example:

```
UPDATE PRODUCT
SET PROD_QOH = PROD_QOH_100
WHERE PROD_CODE = '345TYX';

UPDATE ACCREC
SET AR_BALANCE = AR_BALANCE + 3500
WHERE AR_NUM = '60120010';
COMMIT;
```

Concurrency Control

The coordination of simultaneous execution of transactions in a multiprocessing database system is known as *concurrency control*. The objective of concurrency control is to ensure the serializability of transaction in a multiuser database environment. Concurrency is important, because the simultaneous execution of transactions over a shared database can create several data integrity and consistency problems. Three main problems are lost updates, uncommitted data and inconsistent retrievals.

Lost Updates

Consider the following two concurrent transactions where PROD_QOH represents a particular PRODUCT's quantity on hand. (PROD_QOH is an attribute in the Product table)

Assume the current PROD_QOH value for the product concerned is 35.

Table 1

Transaction	Computation
T_1: purchase 100 units	PROD_QOH = PROD_QOH + 100
T_2: sell 30 units	PROD_QOH = PROD_QOH – 30

Table 1 shows the serial execution of these transactions under normal circumstances, yielding the correct answer: PROD_QOH = 105. But suppose that a transaction is able to read a product's PROD_QOH value from the table before a previous transaction (using the same product) has been committed. The sequence depicted in Table 2 shows how the cost update problem can arise. Note that the first transaction (T_1) has not yet been committed when the second transaction (T_2) is executed. Therefore T_2 still operates on the value 135 to disk which is promptly over written by T_2. As a result, the addition of 100 units is "lost" during the process.

Uncommitted Data

Data are not committed when two transactions, T_1 and T_2 are executed concurrently and the first transaction (T_1) is rolled back after the second transaction (T_2) has already accessed the uncommitted data, thus violating the isolation property of transaction. Consider the same transactions from T_1 and T_2, from above. However, this time T_1 is rolled back to eliminate the addition of the 100 units. Because T_2 subtracts 30 from the original 35 units, the correct answer should be 5.

Table 2

	Computation
T_1: purchase 100 units	PROD_QOH = PROD_QOH + 100 (Rolled Back)
T_2: sell 30 units	PROD_QOH = PROD_QOH_30

Table 2 shows how, under normal circumstances, the serial execution of these transactions yield the correct answer. The uncommitted data problem can arise when the ROLLBACK is completed after T_2 has begun its execution.

Inconsistent Retrievals

Inconsistent retrievals occur when a transaction calculates some summary (aggregate) functions over a set of data, while other transactions are updating the data. The problem is that the transaction might read some data before they are changed and other data after they are changed, thereby yielding inconsistent results.

Example:

1. T_1 calculates the total PROD_QOH of the products stored in the PRODUCT table
2. At the same time, T_2 updates the PROD_QOH for two of the PRODUCT table's products (T_2 represents the correction of a typing error: the user added 30 units to product 345TYX's PROD_QOH but meant to add the 30 units to 125TYZ's PROD_QOH to correct the problem, the user subtracts 30 from product 345TYX's PROD_QOH and adds 30 to product 125TYZ's PROD_QOH).

The computed answer 485 is obviously wrong, because we know the correct answer to be 455.

Transaction Processing Systems

Transaction processing systems are systems with large databases and hundreds of concurrent users that are executing database transactions. For example, banking, credit card processing, stock markets, supermarket checkout, etc.

They require high availability and fast response time for hundreds of concurrent users.

1. A transaction includes one or more database access operations. These can include insertion, deletion, modification, or retrieval operations.
2. *Basic operations*: The basic database access operations that a transaction can include are as follows:
   - read_item (*X*): Reads a database item named *X* into a program variable.
   - Write_item (*X*): Writes the value of program variable *X* into the database item named *X*

Executing a read_item (*X*): Command includes the following steps:

1. Find the address of the disk block that contains item *X*
2. Copy that disk block into a buffer in main memory (if that disk block is not in main memory buffer).
3. Copy item *X* from the buffer to the program variable named x

Executing a write_item (*X*) command includes the following steps:

1. Find the address of the disk block that contains item *X*
2. Copy that disk block into a buffer in main memory (if that disk block is not in main memory buffer)
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.

Step 4 actually updates the database on disk.

The decision about when to store back a modified disk block that is in a main memory buffer is handled by the recovery manager of the DBMS in cooperation with the underlying operating system.

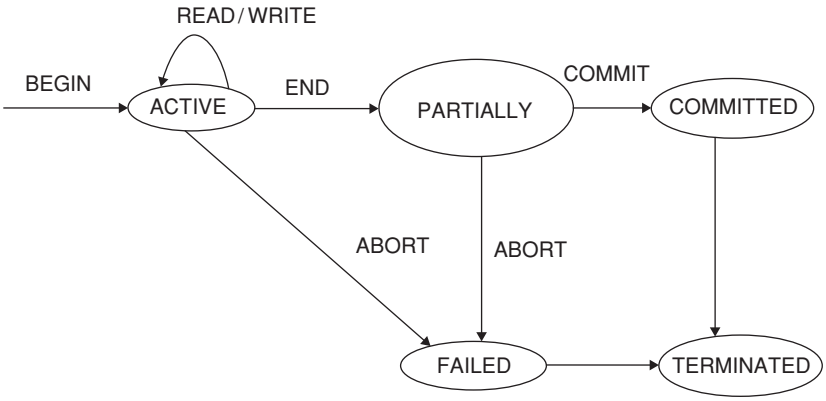


Figure 1 Transactions execution state transition diagram.

For the purpose of recovery, the system needs to keep track of when the transaction starts, terminates, and commits or aborts. Hence, the recovery manager keeps track of the following operations:

1. BEGIN_TRANSACTION: It shows the beginning of Execution of a transaction.
2. READ/WRITE: These specify read or write operations on the database items.
3. END_TRANSACTION: This specifies that READ and WRITE operations have ended and marks the end of transaction execution.
4. COMMIT_TRANSACTION: This shows a successful end of the transaction so that any changes executed by the transaction can be safely committed to the database and will not be undone.
5. ROLL BACK OR ABORT: This shows that the transaction has ended unsuccessfully, so that any changes or effects that the transaction may have applied to the database must be undone.
6. ACTIVE STATE: A transaction goes into an active state immediately after it starts execution where it can issue READ and WRITE operations.
7. PARTIALLY COMMITTED: When the transaction ends, it moves to the partially committed state
8. COMMIT: A transaction reaches its commit point when all its operations that access the database have been executed successfully, and the effect of all the transaction operations on the database have been recorded in the log. Beyond the commit point, the transaction is said to be committed, and its effect is assumed to be permanently recorded in the database.
9. FAILED_STATE: A transaction can go to the failed state 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 operations on the database.
10. TERMINATIED: The terminated state corresponds to the transactions leaving the system.

Concurrency Control with Locking Methods

A lock guarantees exclusive use of a data item to a transaction. In general, if transaction T_1 holds a lock on a data item (e.g., an employee's salary) then transaction T_2 does not have access to that data item. A transaction acquires a lock prior to data access; the lock is released (unlocked) when the transaction is completed, so that another transaction can lock the data item for its exclusive use. All lock information is managed by a lock manager, which is responsible for assigning and policing the locks used by the transactions.

Lock Granularity

Lock granularity indicates the level of lock use. Locking can take place at the following levels: database level, table level, page level, row level and field (or attribute) level.

Database Level

In a database-level lock, the entire database is locked, thus preventing the use of any tables in the database by transaction T_2 while transaction T_1 is being executed. Transaction T_1 and T_2 cannot access the database concurrently, even if they use different tables. This level of locking is suitable for batch processes, but it is not unsuitable for online multiuser DBMSs.

Table Level

In a table-level lock, the entire table is locked, preventing access to any row by transaction T_2 while transaction T_1 is using Table 2 transactions can access the same database, as long as they access different tables. Transactions T_1 and T_2 cannot access the same table even if they try to use different rows, T_2 must wait until T_1 unlocks the table.

Page level

In a page level lock, the DBMS will lock an entire disk page (a disk page or page is the equivalent of a disk block, which can

be described as a (referenced) section of a disk). Transactions T_1 and T_2 access the same table while locking different disk pages. If T_2 requires the use of a row located on a page that is locked by T_1, T_2 must wait until the page is unlocked by T_1.

Row level

The row-level lock is much less restrictive than the locks discussed earlier. The DBMS allows concurrent transactions to access different rows of the same table, even if the rows are located on the same page. A lock exists for each row in each table of the database.

Field level

The field-level lock allows concurrent transactions to access same row as long as they require the use of different fields (attributes) within the row. Although, field-level locking clearly yields the most flexible multi user data access, it requires a high level of computer over head.

Lock Types

1. Binary locks
2. Shared/Exclusive locks

Binary Locks

A binary lock has only two states: locked (1) or unlocked (0). If an object, that is, a database, table, page, or row is locked by a transaction, no other transaction can use that object. If an object is unlocked, any transaction can lock the object for its use. As a rule, a transaction must unlock the object after its termination. Every database operation requires that the affected object be locked. Therefore, every transaction requires a lock and unlock operation for each data item that is accessed. Such operations are automatically scheduled by the DBMS, the user need not concerned about locking or unlocking data items. Binary locks are now considered too restrictive to yield optimal concurrency conditions. For example if two transaction want to read the same database object, the DBMS will not allow this to happen, even though neither transaction updates the database (and therefore, no concurrency problems can occur) concurrency conflicts occur only when two transactions execute concurrently and one of them updates the database.

Shared/Exclusive locks

The tables "shared' and "exclusive" indicate the nature of the lock. The following table comparatively explains both locks.

Exclusive Locks	Shared Locks
An *exclusive lock* exists when access is specifically reserved for the transaction that locked the object.	A *shared lock* exists when concurrent transactions are granted READ access on the basis of a common lock.
The exclusive lock must be used when the potential for conflict exists.	A shared lock produces no conflict as long as the concurrent transactions are read only.
(An exclusive lock is issued when a transaction wants to write (update) a data item and no locks are currently held on that data item by any other transaction.	A shared lock is issued when a transaction wants to read data from the database and no exclusive lock is held on that data item.

Using the shared/exclusive locking concept, a lock can have three states: unlocked, shared (READ) and exclusive (WRITE). 2 READ transactions can be safely executed and shared locks allow several READ transactions to concurrently read the same data item. For example, if transaction T_1 has a shared lock on data item X, and transaction T_2 wants to read data item X, T_2 may also obtain a shared lock on data item X.

If transaction T_2 updates data item X, then an exclusive lock is required by T_2 over data item X. The exclusive lock is granted if and only if no other locks are held on the data item. Therefore, if a shared or exclusive lock is already held on data item X by transaction T_1, an exclusive lock cannot be granted to transaction T_2.

Potential problems with locks

Although locks prevent serious data inconsistencies, their use may lead to two major problems:

1. The resulting transaction schedule may not be serializable.
2. The schedule may create deadlocks. Database *deadlocks* are the equivalent of a traffic gridlock in a big city and are caused when *two transactions wait for each other to unlock data.*

Both problems can be solved. Serializability is guaranteed through a locking protocol known as two-phase locking and deadlocks can be eliminated by using deadlock detection, and prevention techniques. We shall examine these techniques next.

Two-phase Locking to Ensure Serializability

The *two-phase locking protocol* defines how transactions acquire and relinquish locks. It guarantees serializability, but it does NOT prevent deadlocks. The two phases are as follows:

1. A *growing phase*, in which a transaction acquires all the required locks without unlocking any data. Once all locks have been acquired, the transaction is in its locked point.
2. A *shrinking phase*, in which a transaction releases all locks and cannot obtain any new lock.

The two-phase locking protocol is governed by the following rules"

1. Two transactions cannot have conflicting locks.
2. No unlock operation can precede a lock operation in the same transaction.
3. No data are affected until all locks are obtained, that is, until the transaction is in its locked point.

Deadlocks

Deadlocks exist when two transactions, T_1 and T_2, exist in the following mode:

1. T_1 would like to access data item X and then data item Y. (So far, T_1 has locked data item X and T_1 is in progress, it will eventually require to lock data item Y.)
2. T_2 needs to access data items X and Y, to begin. (So far, T_2 has locked data item Y.)

If T_1 has not unlocked data item X, T_2 cannot begin; if T_2 has not unlocked data item Y, T_1 cannot continue. Consequently, T_1 and T_2 wait indefinitely, each waiting for the other to unlock the required data item. Such in a real-world DBMS, many transactions can be executed simultaneously, thereby increasing the probability of generating deadlocks. Note that deadlocks are possible only if one of the transactions wants to obtain an exclusive lock on a data item; no deadlock condition can exist among shared locks.

Three basic techniques exist to control deadlocks:

1. *Deadlock prevention*: A transaction requesting a new lock is aborted if there is a possibility that a deadlock can occur. If the transaction is aborted, all the changes made by this transaction are ROLLED BACK, and all locks obtained by the transaction are released. The transaction is then rescheduled for execution. Deadlock prevention works because it avoids the conditions that lead to deadlocking.
2. *Deadlock detection*: The DBMS periodically tests the database for deadlocks. If a deadlock is found, one of the transactions (the "victim") is aborted (ROLLED BACK and restarted), and the other transaction continues.
3. *Deadlock avoidance*: The transaction must obtain all the locks it needs before it can be executed.

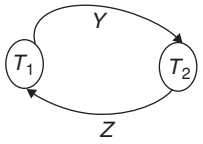
The best deadlock control method depends on the database environment. For example, if the probability of deadlocks is low, deadlock detection is recommended. However, if the probability of deadlocks is high, deadlock prevention is recommended. If response time is not high on the system priority list, deadlock avoidance might be employed.

Deadlock occurs when each transaction T in a set of two or more tractions is waiting for some item that is locked by some other transaction T^1 in the set. Each transaction in the set is on a waiting queue, waiting for one of the other transactions in the set to release the lock on an item

Example:

T_1	T_2
Write lock (z)	
	Read lock (z)
	Read lock (y)
Write lock (y)	

Transaction T_1 is waiting for Y which is locked by Transaction T_2 and transaction T_2 is waiting for z which is locked by transaction T_1. The below graph is called *wait for graph*.



Deadlock Prevention

There are number of deadlock prevention schemes that make a decision about what to do with a transaction involved in a possible deadlock situation:

1. Should it be blocked and made to wait
2. Should it be aborted
3. Should the transaction pre-empt and abort another transaction.

Concurrency Control with Time Stamping Methods

The *time stamping* approach to scheduling concurrent transactions assigns a global unique time stamp to each transaction. The time stamp value produces an explicit order in which transactions are submitted to the DBMS. Time stamps must have two properties: uniqueness and monotonicity. *Uniqueness* ensures that no equal time stamp values can exist, and *monotonicity* ensures that time increases.

All database operations (READ and WRITE) within the same transaction must have the same time stamp. The DBMS executes conflicting operations in time stamp order, thereby ensuring serializability of the transactions. If two transactions conflict, one often is stopped, rescheduled, and assigned a new time stamp value.

The concept of transaction time stamp $TS(T)$, which is a unique identifier assigned to each transaction. The time stamps are based on the order in which transactions start. If transaction T_1 starts before transaction T_2, then $TS(T_1) < TS(T_2)$

1. Older transaction will have the smaller time stamp value
2. Two schemes that prevent deadlock are
   - Wait-die
   - Wound-wait

Suppose that transaction T_k tries to lock an item x but is not able to because x is locked by some other transaction T_L with a conflicting lock.

The rules followed by these schemes are as follows:

1. Wait-die: If T_s (T_k) < T_s (T_L), then (T_k older than T_L) T_k is allowed to wait; otherwise (T_k younger than T_L) abort T_k and restart it later with the same time stamp
2. Wound-wait: If T_s (T_k) < T_s (T_L) then (T_k older than T_L) abort T_L and restart it later with the same time stamp; otherwise (T_k younger than T_L) T_k is allowed to wait

Both schemes end up aborting the younger of the two transactions that may be involved in a deadlock

Concurrency Control with Optimistic Methods

Optimistic methods are based on the assumption that the majority of the database operations do not conflict. A transaction is executed without restrictions until it is committed. Each transaction moves through *two* or *three* phases:

Read phase: The transaction reads the database, executes the needed computations, and makes the updates to a private copy of the database values.

Validation phase: The transaction is validated to assure that the changes made will not affect the integrity and consistency of the database.

If the validation test is positive, transaction goes to the Write Phase.

If the validation test is negative, transaction is restarted, and changes are discarded.

Write phase: The changes are permanently applied to the database.

Serializability

Serializabilty is accepted as 'criterion for correctness' for the interleaved execution of a set of transactions, such an execution is considered to be correct if and only if it is serializable.

1. A set of transactions is serializable if and only if it is equivalent to some serial execution of the same transactions
2. A serial execution is one in which the transactions are run one at a time in some sequence

Schedule: Given a set of transactions, any execution of those transactions interleaved or otherwise is called a *schedule*.

1. Executing the transactions one at a time, with no interleaving constitutes a serial schedule. A schedule that is not serial is an interleaved schedule (or) non-serial schedule.
2. Two schedules are said to be equivalent if and only if they are guaranteed to produce the same result as each other. Thus, a schedule is serializable, and correct, if and only if it is equivalent to some serial schedule.

Two-phase Locking Theorem

If all transactions obey the two phase locking protocol, then all possible interleaved schedule are serializable.

1. Before operating on any object (it could be a database tuple), a transaction must acquire a lock on the object
2. After releasing a lock, a transaction must never go on to acquire any more locks.

A transaction that obeys this protocol thus has two phases: a lock acquisition or "growing phase and a lock releasing or "shrinking" phase

Let '*I*' be an interleaved schedule involving some set of transactions T, T_2, T_3, … T_n.

If '*I*' is serializable, then there exists at least one serial schedule '*S*' involving T_1, T_2, … T_n such that '*I*' is equivalent to '*S*' is said to be a serialization of '*I*'

Let T_i and T_j be any two distinct transactions in the set T_1, T_2, T_3, … T_n. Let T_i precede T_j in the serialization '*S*'. In the interleaved schedule *I*, then the effect must be as if T_i really did execute before T_j. In other words, if *A* and *B* are any two transactions involved is some serializable schedule, then either A logically precedes *B* or *B* logically precedes *A* in that schedule, that is, either *B* can see A's output or *A* can see B's. If the effect is not as if either A ran before *B* or *B* ran before A, then the schedule is not serializable and not correct.

1. A schedule '*S*' of '*n*' transactions T_1, T_2 … T_n is an ordering of the operations of the transactions subject to the constraint that, for each transaction T_i that participates in '*S*', the same order in which they occur in T_i.
2. For the purpose of recovery and concurrency control, we are mainly interested in the 'read item' and 'write item' operations of the transactions, as well as the COMMIT and ABORT operations. A shorthand notation for describing a schedule uses the symbols, '*R*', '*W*', '*C*' and 'A' for the operations read item, write item, commit, and abort respectively, and appends as subscript the transition-id (transaction number) to each operation in the schedule

Example: The schedule of the given set of transactions can be written as follows:

T_1	T_2
Read item (x);	
$X = X - N$;	
	Read item (x);
	$X = X + M$;
Write item (x)	
Read item (y)	
	Write item (x)
	Write item (y)
$Y = Y + N$	
Write item (y)	Commit

Schedule:

$$S: R_1(X); R_2(X); W_1(X); R_1(Y); W_2(X); W_2(Y); W_1(Y); C_2$$

Conflicts: Two operations in a schedule are said to have conflict if they satisfy all three conditions, if

1. they belong to different transactions
2. they access the same data item
3. at least one of the operations is a write item

For example, In the schedule '*S*' given above, the operations $r_1(x)$ and $w_2(x)$ conflict, as do

The operations $r_2(x)$ and $w_1(x)$ and the operations $w_1(x)$ and $w_2(x)$. However the operations $r_1(x)$ and $r_2(x)$ do not conflict, since they are both read operations;

The operations $w_1(x)$ and $w_2(y)$ do not conflict because they operate on distinct data items x and y. The operations $r_1(x)$ and $w_1(x)$ do not conflict, because they belong to the same transaction.

Complete schedule: A schedule *S* of '*n*' transactions T_1, T_2, T_3 … T_n is said to be a complete schedule if the following conditions hold.

1. The operations is '*S*' are exactly those operations in T_1, T_2, … T_n including a commit or abort operation as the last operation for each transaction in the schedule.
2. For any pair of operations from the same transaction T_i, their order of appearance in '*S*' is the same as their order of appearance in T_i
3. For any two conflicting operations, one of the two must occur before the other in the schedule.

The preceding condition (3) allows for two non-conflicting operations to occur in the schedule without defining which occurs first, thus leading to the definition of a schedule as a partial order of the operations in the '*n*' transactions.

It is difficult to encounter complete schedules in a transaction processing system, because new transactions are continually being submitted to the system. Hence, it is useful to define the concept of the 'committed projection *C*(*S*) of schedule *S*; which include only the operations in *S* that belong to committed transactions, that is, transaction T_i whose commit operation is C_i.

Recoverability

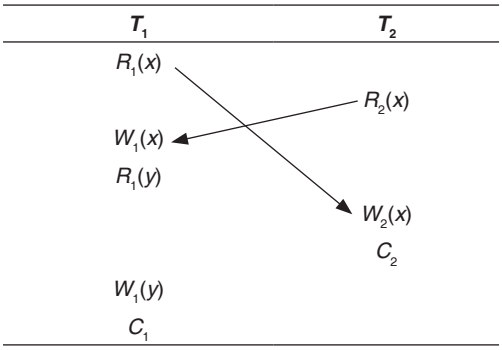
Recoverability ensures that once a transaction *T* is committed, it should never be necessary to roll back *T*. The schedules that theoretically meet this criterion are called *recoverable schedules* and those that do not are called *non-recoverable*, and hence should not be permitted

A schedule '*S*' is recoverable if no transaction *T* in '*S*' commits until all transactions T^1 that have written an item that *T* reads have committed

A transaction *T* reads from transaction T^1 in a schedule *S* if some item *x* is first written by T^1 and later read by *T*. In addition, T^1 should not have been aborted before *T* reads item *x*, and there should be no transactions that write *x* after T^1 writes it and before *T* reads it (unless those transactions, if any, have aborted before *T* reads *x*)

Example:
Consider the given schedule, check whether it is recoverable or not:
S: $R_1(X); R_2(X); W_1(X); R_1(Y); W_2(X); C_2, W_1(Y); C_1$?

Solution:
The given schedule can also be represented as follows:



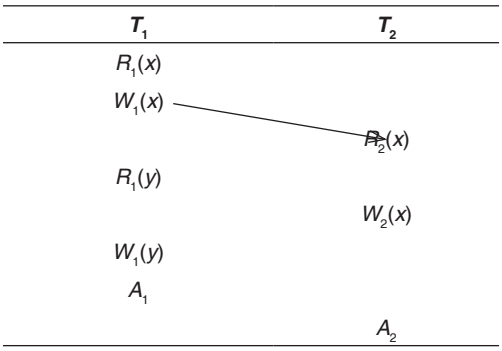
There are two *WR* conflicts, if the schedule consists of *RW* conflict, then we may say that the schedule is not recoverable (if the transaction which is performing read operation commits first)

Cascadeless Schedule

In a recoverable schedule, no committed transaction ever needs to be rolled back. It is possible for a phenomenon known as cascading rollback to occur, when an uncommitted transaction has to be rolled back because it read an item from a transaction that failed.

This is illustrated in the following schedule:

Example:
S: $R_1(X); W_1(X); R_2(X); R_1(Y); W_2(X); W_1(Y); A_1, A_2$
The above schedule is represented as follows:



Transaction T_2 has to be rolled back because it reads item *x* from T_1, and T_1 is then aborted, because cascading rollback

can be quite time consuming since numerous transactions can be rolled back. It is important to characterize the schedules where this phenomenon is guaranteed not to occur.

A scheduled is said to be cascadeless if every transaction in the schedule reads only items that were written by committed transaction. In this case, all items read will not be discarded, so no cascading rollback will occur.

Strict Schedule

A schedule is called *strict schedule*, in which transactions can neither read nor write an item x until the last transaction that wrote x has committed or aborted

1. All strict schedules are cascadeless
2. All cascadeless schedules are recoverable

Equivalence of Schedules

There are several ways to define equivalence of schedules as follows:

1. Result equivalent
2. Conflict equivalent
3. View equivalent

Result Equivalent

Two schedules are called *result equivalent* if they produce the same final state of the database. However, two different schedules may accidentally produce the same final state.

Example: Check whether the two schedules are result equivalent or not:

S_1	S_2
Read item (x);	Read item (x)
$X = X + 20$;	$X = X * 1.1$;
Write item (x);	Write item (x);

Solution:

Schedules S_1 and S_2 will produce the same final database state if they execute on a database with an initial value of $x = 200$; but for other initial values of x, the schedules are not result equivalent

For two schedules to be equivalent, the operations applied to each data item affected by the schedules should be applied to that item in both schedules in the same order. The other two definitions of equivalence of schedules generally used are conflict equivalence and view equivalence

Conflict Equivalence

Two schedules are said to be conflict equivalent if the order of any two conflicting operations is the same in both schedules. If two conflicting operations are applied in different orders in two schedules, the effect can be different on the database or on other transactions in the schedule, and hence the schedules are not conflict equivalent.

Example:

S_1
$r_1(x)$
$W_2(x)$

S_2
$W_2(x)$
$r_1(x)$

S_3
$W_1(x)$
$W_2(x)$

S_4
$W_2(x)$
$W_1(x)$

The value read by $r_1(x)$ can be different in the two schedules. Similarly, if two write operations occur in the order $w_1(x)$, $w_2(x)$ in s_3, and in the reverse order $w_2(x)$, $w_1(x)$ in s_4, the next $r(x)$ operation in the two schedules will read potentially different values.

Testing for Conflict Serializability

The following algorithm can be used to test a schedule for conflict serializability. The algorithm takes read item and write item operations in a schedule to construct a precedence graph or serialization graph, which is a directed graph $G(N, E)$ here N is a set of Nodes $N = \{T_1, T_2, \ldots. T_n\}$ and E is a set of directed edges $E = \{e_1, e_2, \ldots.. E_m\}$ There is one node in the graph for each transaction. T_i in the schedule. Each edge e_i in the graph is of the form $(T_j \rightarrow T_k)$, $1 \text{″} j \text{″} n$, $1 \text{″} k \text{″} n$,

Where T_j is the starting node of e_i and T_k is the ending node of e_i.

Edge is created if one of the operations in T_j appears in the schedule before some conflicting operation in T_k

Algorithm

1. For each transaction T_i participating in schedule S, create a node labelled T_i in the precedence graph
2. For each case in S where T_j executes a read item (x) after T_i executes a write item (x), create an edge $(T_i \rightarrow T_j)$ in the graph
3. For each case in S where T_j executes a write item (x) after T_i executes a read item (x), create an edge $(T_i \rightarrow T_j)$ in the graph
4. For each case in S where T_j executes a write item (x) after T_i executes a write item (x), create an edge $(T_i \rightarrow T_j)$ in the precedence graph
5. The schedule S is serializable, if the precedence graph contains no cycles

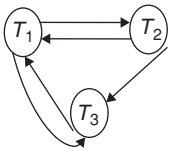
A cycle in a directed graph is a sequence of edges $C = ((T_j \rightarrow T_k), (T_k \rightarrow T_p), \ldots (T_i \rightarrow T_j))$

With the property that the starting node of each edge, except the first edge is the same as the ending node of the previous edge, and the starting node of the first edge is the same as the ending node of the last edge.

Example: Check whether the given schedule is conflict serializable or not by drawing precedence graph:

T_1	T_2	T_3
$R_1(x)$		
	$W_2(x)$	
		$R_3(x)$
$W_1(x)$		
		$W_3(x)$
$R_1(x)$		

Solution:
First identify the conflicts:
$(T_1 \rightarrow T_2)$ WR conflict
$(T_2 \rightarrow T_1)$ WW conflict
$(T_2 \rightarrow T_3)$ RW conflict
$(T_3 \rightarrow T_1)$ WR conflict
$(T_1 \rightarrow T_3)$ WW conflict
Take transactions as nodes in the precedence graph:



The precedence graph has cycle, which says that the schedule is not serializable.

View Equivalence and View Serializability

View equivalence is less restrictive compared to conflict equivalence. Two schedules S and S' are said to be view equivalent if the following three conditions hold:

1. The same set of transactions participate in S and S', and S and S' include the same operations of those transactions
2. For any operation $r_i(x)$ of T_i in S, if the value of x read by the operation has been written by an operation $w_j(x)$ of T_j, the same condition must hold for the value of x read by operation $r_i(x)$ of T_j in S'
3. If the operation $w_k(y)$ of T_k is the last operation to write Y in S, then $W_k(y)$ of T_k must also be the last operation to write item Y in S'

The idea behind view equivalence is that as long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same result. Hence the read operation is said to see the same view in both schedules:

1. A schedule S is said to be view serializable if it is view equivalent to a serial schedule.
2. All conflict serializable schedules are view serializable, but vice versa is not true.

Exercises

Practice Problem 1

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Consider the given schedules S_1 and S_2
S_1: $r_1(x), r_1(y), r_2(x), r_2(y), w_2(y), w_1(x)$
S_2: $r_1(x), r_2(x), r_2(y), w_2(y), r_1(y), w_1(x)$
Which schedule is conflict serializable?
(A) S_1 (B) S_2
(C) S_1 and S_2 (D) None of these

2. Consider the given schedule with three transactions T_1, T_2 and T_3:

T_1	T_2	T_3
$r_1(x)$		
	$r_2(y)$	
		$r_3(y)$
	$w_2(y)$	
$w_1(x)$		
		$w_3(x)$
	$r_2(x)$	
	$w_2(x)$	

Which of the following is correct serialization?
(A) $T_2 \rightarrow T_1 \rightarrow T_3$ (B) $T_1 \rightarrow T_3 \rightarrow T_2$
(C) $T_3 \rightarrow T_1 \rightarrow T_2$ (D) None of these

3. Consider the three data items D_1, D_2 and D_3 and the following execution of schedules of transactions T_1, T_2 and T_3:

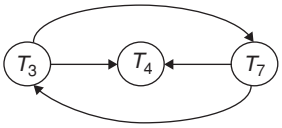
T_1	T_2	T_3
	$R(D_2)$	
	$R(D_2)$	
	$W(D_2)$	
		$R(D_2)$
		$R(D_3)$
$R(D_1)$		
$W(D_1)$		
		$W(D_2)$
		$W(D_3)$
	$R(D_1)$	
$R(D_2)$		
$W(D_2)$		
	$W(D_1)$	

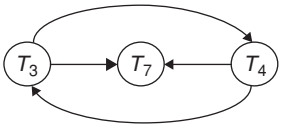
Which of the following is true?
(A) The schedule is conflict serializable
(B) The schedule is not conflict serializable
(C) The schedule has deadlock
(D) Both (A) and (C)

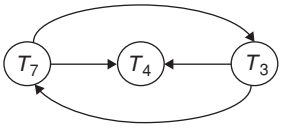
4. Consider the given schedule

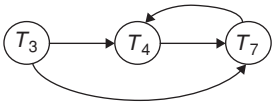
T_3	T_4	T_7
$R(Q)$		
	$W(Q)$	
$W(Q)$		
		$R(Q)$
		$W(Q)$

Which of the following is the correct precedence graph for the above schedule?

(A) 

(B) 

(C) 

(D) 

5. Consider two Transactions T_1 and T_2 and four schedules: S_1, S_2, S_3 and S_4 of T_1 and T_2:

T_1: $r_1(x), w_1(x), w_1(y)$

T_2: $r_2(x), r_2(y), w_2(y)$

S_1: $r_1(x), r_2(x), r_2(y), w_1(x), w_1(y), w_2(y)$

S_2: $r_1(x), r_2(x), r_2(y), w_1(x), w_2(y), w_1(y)$

S_3: $r_1(x), w_1(x), r_2(x), w_1(y), r_2(y), w_2(y)$

S_4: $r_2(x), r_2(y), r_1(x), w_1(x), w_1(y), w_2(y)$

Which schedules are conflict serializable in the given schedules?

(A) S_1 and S_2
(B) S_1 and S_3
(C) S_2 and S_3
(D) S_1 and S_4

6. Consider the following transactions with data items P and Q initialized to '0':

T_1: read(P)
Read(Q)
if $p = 0$ then $Q = Q + 1$
Write(Q)

T_2: read(Q)
Read(P)
if $Q = 0$ then $p = p + 1$
Write (P)

Any non-serial interleaving of T_1 and T_2 for concurrent execution leads to

(A) a serializable schedule
(B) a schedule that is not conflict serializable
(C) a conflict serializable schedule
(D) a schedule for which a precedence graph cannot be drawn

7. Consider the concurrent execution of two transactions T_1 and T_2, if the initial values of x, y, M and N are 200, 100, 10, 20 respectively. What are the final values of x and y?

T_1	T_2
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)	

(A) 220, 110 (B) 210, 120
(C) 220, 120 (D) 210, 110

8. For the above data, if the transactions are executed in serial manner, what would be the values of X and Y at the end of the serial execution of T_1 and T_2?

T_1	T_2
Read-item(x)	
$X = X - N$	
Write-item(x)	
Read-item(y)	
$Y = Y + N$	
Write-item(y)	
	Read-item(x)
	$X = X + M$
	Write-item(x)

(A) 190, 120
(B) 180, 120
(C) 190, 110
(D) 180, 110

9. Consider the given two transactions T_1 and T_2:

T_1: $r_1(x), w_1(x), r_1(y)$

T_2: $r_2(x), r_2(y), w_2(x), w_2(y)$

Which of the following schedules are complete schedules?

(A) $r_1(x), r_2(x), w_1(x), r_1(y), r_2(y), w_2(x), w_2(y)$
(B) $r_2(x), r_1(x), r_2(y), w_1(x), w_2(x), r_1(y), w_2(y)$
(C) $r_1(x), r_1(y), r_2(x), r_2(y), w_1(x), w_2(x), w_2(y)$
(D) All the above

10. Consider the given schedule with data-locks on data-items, check whether it has dead-lock or not. The locks are shared-lock(S) and Exclusive-lock(X). Shared-lock is also called Read-lock, Exclusive-lock is also called Write-lock. Read and Write operations are denoted by *R* and *W*, respectively.

T_1	T_2	T_3	T_4
S(A)			
R(A)			
	X(B)		
	W(B)		
S(B)			
		S(C)	
		R(C)	
	X(C)		
			X(B)
		X(A)	

Which of the following is incorrect?

(A) $T_1 \rightarrow T_2$ (B) $T_3 \rightarrow T_1$
(C) $T_2 \rightarrow T_3$ (D) $T_4 \rightarrow T_3$

11. Consider the three transactions T_1, T_2 and T_3 and the schedule S_1 as given below. Draw the serializability (precedence) graph for S_1, and state whether the schedule is serializable or not. If a schedule is serializable, which one of the following is equivalent serial schedule?

T_1: $r_1(x), r_1(z), w_1(x)$
T_2: $r_2(z), r_2(y), w_2(z), w_2(y)$
T_3: $r_3(x), r_3(y), w_3(y)$
S_1: $r_1(x), r_2(z), r_1(z), r_3(x), r_3(y), w_1(x), w_3(y), r_2(y), w_2(z), w_2(y)$?

(A) $r_3(x), r_3(y), w_3(y), r_1(x), r_1(z), w_1(x), r_2(z), r_2(y), w_2(z), w_2(y)$
(B) $r_1(x), r_1(z), w_1(x), r_2(z), r_2(y), w_2(z), w_2(y), r_3(x), r_3(y), w_3(y)$
(C) $r_2(z), r_2(y), w_2(z), w_2(y), r_3(x), r_3(y), w_3(y), r_1(x), r_1(z), w_1(x)$
(D) $r_2(z), r_2(y), w_2(z), w_2(y), r_1(x), r_1(z), w_1(x), r_3(x), r_3(y), w_3(y)$

12. Consider the data given in the above question. Draw the precedence graph for S_2 and state whether each schedule is serializable or not. If a schedule is serializable, which of the following is equivalent serial schedule?

S_2: $r_1(x), r_2(z), r_3(x), r_1(z), r_2(y), r_3(y), w_1(x), w_2(z), w_3(y), w_2(y)$

(A) $r_3(x), r_3(y), w_3(y), r_1(x), r_1(z), w_1(x), r_2(z), r_2(y), w_2(z), w_2(y)$
(B) $r_1(x), r_1(z), w_1(x), r_2(z), r_2(y), w_2(z), w_2(y), r_3(x), r_3(y), w_3(y)$
(C) $r_2(z), r_2(y), w_2(z), w_2(y), r_3(x), r_3(y), w_3(y), r_1(x), r_1(z), w_1(x)$
(D) $r_2(z), r_2(y), w_2(z), w_2(y), r_1(x), r_1(z), w_1(x), r_3(x), r_3(y), w_3(y)$

13. Consider schedule S_3, which is a combination of transactions T_1, T_2 and T_3 from Q. No.11.

S_3: $r_1(x), r_2(z), r_1(z), r_3(x), r_3(y), w_1(x), c_1, w_3(y), c_3, r_2(y), w_2(z), w_2(y), c_2$?

Which of the following is true?

(A) Recoverable and conflict serializable
(B) Recoverable but not conflict serializable
(C) Conflict serializable but not Recoverable
(D) Not recoverable and not conflict serializable

14. Consider the given schedule:

S_4: $r_1(x), r_2(z), r_1(z), r_3(x), r_3(y), w_1(x), w_3(y), r_2(y), w_2(z), w_2(y), c_1, c_2, c_3$:

Which of the following is true?

(A) Recoverable and conflict serializable
(B) Recoverable but not conflict serializable
(C) Conflict serializable but not Recoverable
(D) Not recoverable and not conflict serializable

15. Which of the following is correct for the below compatibility matrix?

Mode of Locks Currently held by other transactions		**Shared-Lock**	**Exclusive-Lock**
	S		
	X		

S – shared - Lock, *X* – Exclusive - Lock

(A)

	S	*X*
S	No	No
X	Yes	No

(B)

	S	*X*
S	Yes	No
X	No	No

(C)

	S	*X*
S	Yes	Yes
X	No	No

(D)

	S	*X*
S	No	Yes
X	No	No

16. Consider the following schedule with locking:

T_1	T_2
Lock – $X(A)$	
$R(A)$	
$W(A)$	
	Lock – $X(B)$
	$R(B)$
	$W(B)$
	Lock – $X(A)$
Lock – $X(B)$	

Which of the following is true?
(A) schedule is in Dead–Lock state
(B) schedule is conflict serializable
(C) schedule is not conflict serializable
(D) Both A and B

17. Consider the given set of transactions:

T_1	T_2
	SELECT AVG (balance) FROM Account
INSERT INTO Account VALUES (487, 2000); COMMIT	
	SELECT AVG (balance) FROM Account COMMIT

The above problem is a case of
(A) READ UNCOMMITTED
(B) RAD COMMITTED
(C) REPEATABLE READ
(D) DIRTY READ

18. Consider the given set of transactions

T_1	T_2
UPDATE ACCOUNT SET balance = balance – 1000 WHERE number = 586;	
	SELECT AVG (balance) FROM Account
ROLL BACK	
	COMMIT

The above problem is a case of
(A) READ UNCOMMITTED
(B) READ COMMITTED
(C) DIRTY READ
(D) BOTH A and C

19. Consider the following set of transactions

T_1	T_2
	SELECT AVG (balance) FROM Account
UPDATE Account SET balance = balance – 4000 WHERE number = 586; COMMIT	
	SELECT AVG (balance) FROM Account COMMIT

The above problem is a case of
(A) READ UNCOMMITTED
(B) READ COMMITTED
(C) REPEATABLE READ
(D) DIRTY READ

20. Consider the following schedule with locks on data items:

T_1	T_2	T_3
$X(A)$		
	$X(A)$	
		$X(A)$
$S(B)$		
	$S(B)$	

Which of the following is incorrect?
(A) $T_2 \rightarrow T_1$
(B) $T_3 \rightarrow T_2$
(C) $T_3 \rightarrow T_1$
(D) $T_1 \rightarrow T_3$

Practice Problem 2

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Which of the following is false with respect to B^+ - trees of order p?
(A) Each internal node has at most p tree pointers.
(B) Each leaf node has at most $\left\lceil \left(\frac{p}{2} \right) \right\rceil$ values.
(C) Each internal node, except the root, has at least $\left\lceil \left(\frac{p}{2} \right) \right\rceil$ tree pointers.
(D) All leaf nodes are at same level.

2. Consider below transactions:

T_1	T_2
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);	

Which of the following problem will occur during the concurrent execution of the above transactions?

(A) Lost update problem because of incorrect X.
(B) Lost update problem because of incorrect Y.
(C) Dirty read problem because of incorrect X.
(D) Dirty read problem because of incorrect Y.

3. Consider the scheduled:

S: $r_1(X)$; $r_2(X)$; $w_1(X)$; $r_1(Y)$; $w_2(X)$; C_2; $w_1(Y)$; C_1;

This schedule is

(A) Recoverable (B) Non-recoverable
(C) Strict schedule (D) Both (A) and (C)

4. Consider below schedule:

T_1	T_2
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);	

This schedule is

(A) Serializable
(B) Not serializable
(C) Under dead lock
(D) Both (B) and (C)

5. Let, current number of file records = r
maximum number of records = bfr
current number of file buckets = N
Then what will be the file load factor?

(A) $\frac{r}{(bfr * N)}$ (B) $r + (bfr * N)$

(C) $r * (bfr * N)$ (D) $r * (bfr + N)$

6. Match the following:

	LIST I		LIST II
1.	Primary index	A.	Ordered key field
2.	Clustering index	B.	Non-ordered field
3.	Secondary index	C.	Ordered non-key field

(A) 1 – A, 2 – B, 3 – C
(B) 1 – A, 2 – C, 3 – B
(C) 1 – C, 2 – B, 3 – A
(D) 1 – C, 2 – A, 3 – B

7. Consider a file with 30,000 fixed length records of size 100 bytes stored on a disk with block size 1024 bytes. Suppose that a secondary index on a non-ordering key field is constructed with key field size 9 bytes and block pointer 6 bytes. What will be the number of blocks needed for the index?

(A) 68 (B) 442
(C) 1500 (D) 3000

8. Match the following:

	Index type		Number of Index entries
1.	Primary Index	A.	Blocks in data file
2.	Clustering index	B.	Record in data file
3.	Secondary key index	C.	Distinct index filed values

(A) 1 – A, 2 – B, 3 – C (B) 1 – A, 2 – C, 3 – B
(C) 1 – C, 2 – B, 3 – A (D) 1 – C, 2 – A, 3 – B

9. Which of the following is true with respect to B – Tree of order p?

(A) Each node has at most p tree pointers.
(B) Each node, except the root and leaf nodes, has at least $\left\lceil \frac{p}{2} \right\rceil$ tree pointers.
(C) All leaf nodes are at the same level.
(D) All of these.

10. What is the amount of unused space in allocation of unspanned fixed records of size R on a block of size B bytes?

(A) $B - R$ (B) $B - \left\lfloor \frac{B}{R} \right\rfloor$

(C) $B - \left(\left\lfloor \frac{B}{R} \right\rfloor * R \right) 4$ (D) $\left(\left\lfloor \frac{B}{R} \right\rfloor * R \right) - B$

11. What is the average time required to access a record in a file consisting of b blocks using unordered heap linear search?

(A) b (B) $b/2$
(C) $\log_2^b$ (D) b^2

12. Consider a file of fixed length records of size R bytes. If the block size is B bytes, then the blocking factor will be

(A) $B \times R$ records (B) $\left\lfloor \frac{B}{R} \right\rfloor$ records

(C) $\left\lceil \frac{B}{R} \right\rceil$ records (D) $B + R$ records

13. Consider the following relation instance:

P	Q	R
1	4	2
1	5	3
1	6	3
3	2	2

Which of the following FDs are satisfied by the instance?

(A) $PQ \rightarrow R$ and $R \rightarrow Q$ (B) $QR \rightarrow P$ and $Q \rightarrow R$

(C) $QR \rightarrow P$ and $P \rightarrow R$ (D) $PR \rightarrow Q$ and $Q \rightarrow P$

14. Consider an ordered file with 30,000 records stored on a disk with block size of 1024 bytes. The records are of fixed size and are of unspanned, with record length 100 bytes. What is the number of accesses required to access a data file using binary search?

(A) 10 (B) 12

(C) 1500 (D) 3000

15. What is the blocking factor for an index if the ordering key field size is 9 bytes and block pointer is 6 bytes long, and the disk block size is 1024 bytes?

(A) 114 (B) 171

(C) 341 (D) 68

16. For a set of n transactions, there exist _____ different valid serial schedules

(A) n (B) n^2

(C) $n/2$ (D) $n!$

17. The number of possible schedules for a set of n transactions is

(A) lesser than $n!$ (B) much larger than $n!$

(C) $n!$ (D) None

18. Which one of the following is conflict operation?

(A) Reads and writes from the same transaction

(B) Reads and writes from different transaction

(C) Reads and writes from different transactions on different data items.

(D) Reads and writes from different transaction on same data.

19. The following schedule S: $r_3(x), r_2(x), w_3(x), r_1(x), w_1(x)$ is conflict equivalent to serial schedule

(A) $T_1 \rightarrow T_3 \rightarrow T_1$ (B) $T_2 \rightarrow T_1 \rightarrow T_3$

(C) $T_1 \rightarrow T_2 \rightarrow T_3$ (D) None

20. The following schedule S: $R_1(x), R_2(x), W_1(x), W_2(x)$ is

(A) Conflict serializable (B) View serializable

(C) Both (D) None

Previous Years' Questions

1. Consider the following four schedules due to three transactions (indicated by the subscript) using *read* and *write* on a data item x, denoted by $r(x)$ and $w(x)$, respectively. Which one of the them is conflict serializable? **[2014]**

(A) $r_1(x); r_2(x); w_1(x); r_3(x); w_2(x)$

(B) $r_2(x); r_1(x); w_2(x); r_3(x); w_1(x)$

(C) $r_3(x); r_2(x); r_1(x); w_2(x); w_1(x)$

(D) $r_2(x); w_2(x); r_3(x); r_1(x); w_1(x)$

2. Consider the following schedule S of transactions T_1, T_2, T_3, T_4:

T_1	T_2	T_3	T_4
	Reads (X)		
		Writes (X)	
		Commit	
Writes (X)			
Commit			
	Writes (Y)		
	Reads (Z)		
	Commit		
			Reads (X)
			Reads (Y)
			Commit

Which one of the following statements is correct? **[2014]**

(A) S is conflict-serializable but not recoverable

(B) S is not conflict-serializable but is recoverable

(C) S is both conflict-serializable and recoverable

(D) S is neither conflict-serializable nor it is recoverable

3. Consider the following transaction involving two bank accounts x and y.

read (x) ; $x := x - 50$; write (x) ; read(y); $y := y + 50$; write(y)

The constraint that the sum of the accounts x and y should remain constant is that of **[2015]**

(A) Atomicity (B) Consistency

(C) Isolation (D) Durability

4. Consider a simple checkpointing protocol and the following set of operations in the log.

(start, T_4); (write, T_4, y, 2, 3); (start, T_1); (commit, T_4); (write, T_1, z, 5, 7);

(checkpoint);

(start, T_2); (write, T_2, x, 1, 9); (commit, T_2); (start, T_3), (write, T_3, z, 7, 2);

If a crash happens now and the system tries to recover using both undo and redo operations. What are the contents of the undo list and the redo list? **[2015]**

(A) Undo: T_3, T_1; Redo: T_2
(B) Undo: T_3, T_1; Redo: T_2, T_4
(C) Undo: none; Redo: T_2, T_4, T_3, T_1
(D) Undo: T_3, T_1, T_4; Redo: T_2

5. Consider the following partial schedule S involving two transactions T_1 and T_2. Only the read and the write operations have been shown. The read operation on data item P is denoted by read(P) and the write operation on data item P is denoted by write(P)

Time Instance	Transaction – id	
	T_1	T_2
1	read(A)	
2	write(A)	
3		read(C)
4		write(C)
5		read(B)
6		write(B)
7		read(A)
8		commit
9	read(B)	

Schedule S

Suppose that the transaction T_1 fails immediately after time instance 9. Which one of the following statements is correct? **[2015]**

(A) T_2 must be aborted and then both T_1 and T_2 must be re-started to ensure transaction atomicity.
(B) Schedule S is non-recoverable and cannot ensure transaction atomicity.
(C) Only T_2 must be aborted and then re-started to ensure transaction atomicity.
(D) Schedule S is recoverable and can ensure atomicity and nothing else needs to be done.

6. Which one of the following is **NOT** a part of the ACID properties of database transactions? **[2016]**

(A) Atomicity
(B) Consistency
(C) Isolation
(D) Deadlock - freedom

7. Consider the following two phase locking protocol. Suppose a transaction T accesses (for read or write operations), a certain set of objects $\{O_1, \ldots, O_k\}$. This is done in the following manner: **[2016]**

Step 1. T acquires exclusive locks to $O_1, \ldots O_k$ in increasing order of their addresses.

Step 2. The required operations are performed.

Step 3. All locks are released.

This protocol will

(A) guarantee serializability and deadlock-freedom.
(B) guarantee neither serializability nor deadlock-freedom.
(C) guarantee serializability but not deadlock-freedom.
(D) guarantee deadlock-freedom but not serializabilty.

8. Suppose a database schedule S involves transactions $T_1, \ldots T_n$. Construct the precedence graph of S with vertices representing the transactions and edges representing the conflicts. If S is serializable, which one of the following orderings of the vertices of the precedence graph is guaranteed to yield a serial schedule? **[2016]**

(A) Topological order
(B) Depth - first order
(C) Breadth - first order
(D) Ascending order of transaction indices

9. Consider the following database schedule with two transactions T_1 and T_2.

$S = r_2(X); r_1(X); r_2(Y); w_1(X); r_1(Y); w_2(X); a_1; a_2$

Where $r_i(Z)$ denotes a *read* operation by transaction T_i on a variable Z, $w_i(Z)$ denotes a *write* operation by T_i on a variable Z and *ai* denotes an *abort* by transaction T_i.

Which one of the following statements about the above schedule is **TRUE**? **[2016]**

(A) S is non - recoverable
(B) S is recoverable, but has a cascading abort
(C) S does not have a cascading abort
(D) S is strict.

10. In a database system, unique timestamps are assigned to each transaction using Lamport's logical clock. Let $TS(T_1)$ and $TS(T_2)$ be the timestamps of transactions T_1 and T_2 respectively. Besides, T_1 holds a lock on the resource R, and T_2 has requested a conflicting lock on the same resource R. The following algorithm is used to prevent deadlocks in the database system assuming that a killed transaction is restarted with the same timestamp.

if $TS(T_2) < TS(T_1)$ then
    T_1 is killed
else T_2 waits.

Assume any transaction that is not killed terminates eventually. Which of the following is TRUE about the database system that uses the above algorithm to prevent deadlocks? **[2017]**

(A) The database system is both deadlock-free and starvation-free.
(B) The database system is deadlock-free, but not starvation-free.

(C) The database system is starvation-free, but not deadlock-free.

(D) The database system is neither deadlock-free nor starvation-free.

11. Two transactions T_1 and T_2 are given as

$$T_1 : r_1(X)w_1(X)r_1(Y)w_1(Y)$$

$$T_2 : r_2(Y)w_2(Y)r_2(Z)w_2(Z)$$

where $r_i(V)$ denotes a *read* operation by transaction T_i on a variable V and $w_i(V)$ denotes a *write* operation by transaction T_i on a variable V. The total number of conflict serializable schedules that can be formed by T_1 and T_2 is _____. **[2017]**

Answer Keys

Practice Problem I

1. B	**2.** B	**3.** B	**4.** B	**5.** C	**6.** B	**7.** B	**8.** A	**9.** A	**10.** D
11. A	**12.** A	**13.** A	**14.** C	**15.** B	**16.** D	**17.** C	**18.** D	**19.** B	**20.** D

Practice Problem I

1. B	**2.** A	**3.** A	**4.** D	**5.** A	**6.** B	**7.** B	**8.** B	**9.** D	**10.** C
11. B	**12.** B	**13.** B	**14.** B	**15.** D	**16.** D	**17.** C	**18.** D	**19.** A	**20.** D

Previous Years' Questions

1. D	**2.** C	**3.** B	**4.** A	**5.** B	**6.** D	**7.** A	**8.** A	**9.** C	**10.** A
11. 54									

Chapter 5

File Management

LEARNING OBJECTIVES

- *Files*
- *Memory hierarchies*
- *Description of disk devices*
- *File records*
- *Sorted files*
- *Hashing techniques*
- *Extendible hashing*
- *Index update*
- *Clustering index*
- *B-Trees*
- *B+Trees*
- *Over flow in internal node*

Files

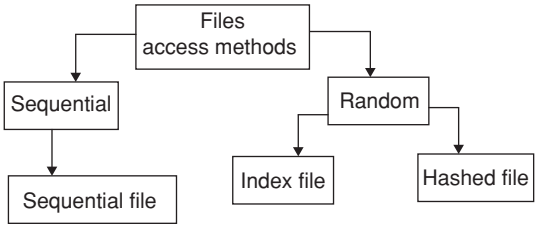
Databases are stored on magnetic disks as files of records. Computer storage media form a storage hierarchy that includes two main categories.

Primary storage This category includes storage media that can be operated on, directly by CPU, such as the computer main memory and cache memory. Primary storage provides fast access but is of limited storage capacity.

Secondary storage This category includes magnetic disks, optical disks, and tapes. These devices usually have a larger capacity, less cost, and slower access to data. Data in secondary storage cannot be processed directly by the CPU, it must be copied into primary storage.

File Structure

Taxonomy of file structure



Sequential file A sequential file is one in which records can only be accessed sequentially, one after another from beginning to end. Records are stored contiguously on the storage device.

Index files These files are used to access a record in the file. The entire index file is loaded into main memory data and indexes are stored in the same file. The term 'index file' is used as a synonym for the term 'database file'. The index file contains parameters that specify the name and location of file used to store DB.

Indexing Indexing mechanism is used to speed up access to desired data. An index file consists of records (called *index entries*) of the form.

Search-key	Pointer

Index files are typically much smaller than the original file.

Ordered indices In ordered index, index entries are stored, sorted on the search-key value.
Example: Author catalogue in library.

Memory Hierarchies

At the primary storage level, the memory hierarchy includes cache memory which is a static RAM.

The next level of primary storage is DRAM (dynamic RAM) which provides the main work area for the CPU for keeping programs and data and is called the *main memory*.

At the secondary storage level, the hierarchy includes magnetic disks, as well as mass storage in the form of CD-ROM (compact disk read-only memory) and tapes. Programs reside in DRAM and large permanent databases reside on secondary storage.

Another form of memory, *flash memory*, is non-volatile. Flash memories are high-density, high-performance memories using EEPROM (electrically erasable programmable read-only

memory) technology. The advantage of flash memory is the fast access speed, the disadvantage is that an entire block must be erased and written over at a time. Finally, magnetic tapes are used for archiving and backup storage of data.

Description of Disk Devices

Magnetic disks are used for storing large amounts of data. The capacity of a disk is the number of bytes it can store. A disk is single sided if it stores information on only one of its surfaces and double sided if both surfaces are used. To increase storage capacity, disks are assembled into a disk pack, which may include many disks and hence many surfaces. Information is stored on a disk surface in concentric circles with small width, each having a distinct diameter. Each circle is called a *track*. For disk packs, the tracks with the same diameter on the various surfaces are called a *cylinder* because of the shape they would form if connected in space.

A track usually contains a large amount of information; it is divided into smaller blocks (or) sectors. The division of track into equal-sized disk blocks (or pages) is set by the operating system during disk formatting.

Blocks are separated by fixed-size inter-block gaps, which include specially coded control information written during disk initialization. This information is used to determine which block on the track follows each inter block gap. Transfer of data between main memory and disk takes place in units of disk blocks. The hardware address of a block is the combination of a cylinder number, track number and block number is supplied to the disk I/O hardware.

The actual hardware mechanism that reads or writes a block is the disk read/write head, which is part of a system called a *disk drive*. A disk is mounted in the disk drive, which includes a motor that rotates the disk. To transfer a disk block, given its address, the disk controller must first mechanically position the read/write head on the correct track. The time required to do this is called the *seek time*. There is another delay called *rotational delay* or *latency*; the beginning of the desired block rotates into position under the read/write head. It depends on the RPM of the disk. Finally, some additional time is needed to transfer the data, which is called *block-transfer time*. Hence, the total time needed to locate and transfer an arbitrary block, given its address is the sum of the seek time, rotational delay and block transfer time. The seek time and rotational delay are usually much larger than the block transfer time.

File Records

Data is usually stored in the form of *records*. Each record consists of a collection of related data values or items where each value is of one or more bytes and corresponds to a particular field of the record. Records describe entities and their attributes.

Record type A collection of field names and their corresponding data types constitutes a record type (or) record format.

A file is a sequence of records. If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records. If different records in the file have different sizes, the file is said to be made up of variable-length records.

Spanned Versus Unspanned Records

The records of a file must be allocated to disk blocks because a block is the unit of data transfer between disk and memory. When the block size is larger than the record size, each block will contain numerous records, although some files may have unusually large records that cannot fit in one block.

Suppose that the block size is B bytes. For a file of fixed-length records of size R bytes, with $B \geq R$, we can fit

$$bfr = \lfloor B/R \rfloor \text{ records per block}$$

The value *bfr* is called the *blocking factor* for the file. Some times R may not divide B exactly, so we have some unused space in each block equal to $B - (bfr * R)$ bytes. To utilize this unused space, we can store part of a record on one block and the rest on another. A pointer at the end of the first block points to the block containing the remainder of the record in case it is not the next consecutive block on disk. This organization is called *spanned*, because records can span more than one block. Whenever a record is larger than a block, we must use a spanned organization. If records are not allowed to cross block boundaries, the organization is called *unspanned*. This is used with fixed-length records having $B > R$, because it makes each record start at a known location in the block. For variable-length records, either a spanned or an unspanned organization can be used.

For variable-length records using spanned organization, each block may store a different number of records. In this case, the blocking factor *bfr* represents the average number of records per block for the file. We can use *bfr* to calculate the number of blocks '*b*' needed for a file of '*r*' records.

$b = \lceil (r/bfr) \rceil$ blocks

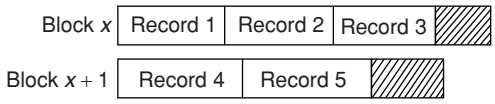


Figure 1 Unspanned records

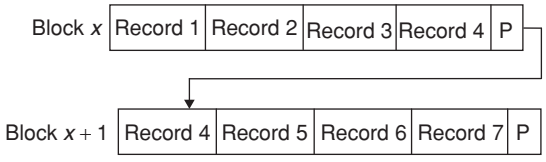


Figure 2 Spanned

There are several standard techniques for allocating the blocks of a file on disk. In contiguous allocation, the file blocks are allocated to consecutive disk blocks. In linked

allocation, each file block contains a pointer to the next file block. A combination of the two allocates clusters of consecutive disk blocks, and the clusters are linked. Clusters are sometimes called file *segments* (or) *extents*. Another possibility is to use indexed allocation, where one or more index blocks contain pointers to the actual file blocks.

Sorted Files (Ordered Records)

We can physically order the records of a file on disk based on the values of the one of their fields called the *ordering field*. This leads to an ordered or sequential file. If the ordering field is also a key field of the file, a field guaranteed to have a unique value in each record, then the field is called the *ordering key for the file*.

Advantages

1. Reading the records in order of the ordering key values becomes extremely efficient, because no sorting is required.
2. Finding the next record from the current one in order of the ordering key usually requires no additional block access, because the next record is in the same block as the current one.
3. Using a search condition based on the value of an ordering key field results in faster access when the binary search technique is used. This constitutes an improvement over linear searches, although it is not often used for disk files.

A binary search for disk files can be done on the blocks rather than on the records. Suppose that a file has 'b' blocks numbered 1, 2 ,…, b, the records are ordered by ascending value of their ordering key field and we are searching for a record whose ordering key field value is K. Assuming that disk addresses of the file blocks are available in the file header, the binary search usually accesses $\log_2^{(b)}$ blocks, whether the record is found (or) not, an improvement over linear searches, where, on the average, (b/2) blocks are accessed when the record is found and 'b' blocks are accessed when the record is not found.

Type of Organization	Access Method	Average Time to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Ordered scan	$b/2$
Ordered	Binary search	$\log_2 b$

Ordered files are rarely used in database applications unless an additional access path, called a *primary index*, is used; this results in an indexed sequential file. This further improves the random access time on the ordering key field.

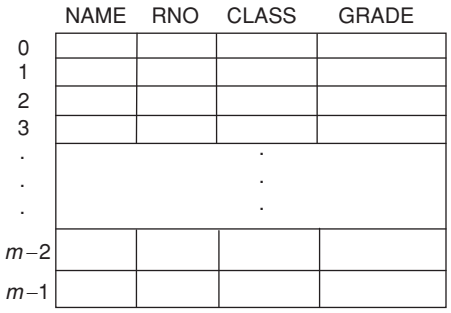
Hashing Techniques

The other type of primary file organization is based on hashing, which provides very fast access to records on certain search conditions. This organization is usually called a *hash file*.

The search condition must be an equality condition on a single field, called the *hash field* of the file. If the hash is also a key field of the file, in which case it is called the *hash key*.

The idea behind hashing is to provide a function 'h', called a hash function or randomizing function, which is applied to the hash field value of a record and yields the address of the disk block in which the record is stored. We need only a single-block access to retrieve that record.

Example:



Internal Hashing

For internal files, hashing is implemented as a hash table through the use of an array of records. Suppose that the array index range is from 0 to $M - 1$, then we have M slots whose addresses corresponds to the array indexes. We choose a hash function that transforms the hash field value into an integer between 0 and $M - 1$. One common hash function is the $h(K) = K \bmod M$ function, which returns the remainder of an integer hash field value K after division by M; this value is then used for the record address.

Non-integer hash field values can be transformed into integers before the mod function is applied. For character strings, the numeric (ASCII) codes associated with characters can be used in the transformation.

A collision occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record. In this situation, we must insert the new record in some other position, since its hash address is occupied. The process of finding another position is called *collision resolution*. There are different methods for collision resolution as follows:

Open addressing Proceeding from the occupied position specified by the hash address, the program checks the subsequent positions in order until an unused (empty) position is found.

Chaining For this method, various overflow locations are kept, usually by extending the array with a number of overflow positions. In addition, a pointer field is added to each record location. A collision is resolved by placing the new record in an unused overflow location and setting the pointer of the occupied hash address location to the address of that overflow location. A linked list of overflow records for each hash address is thus maintained.

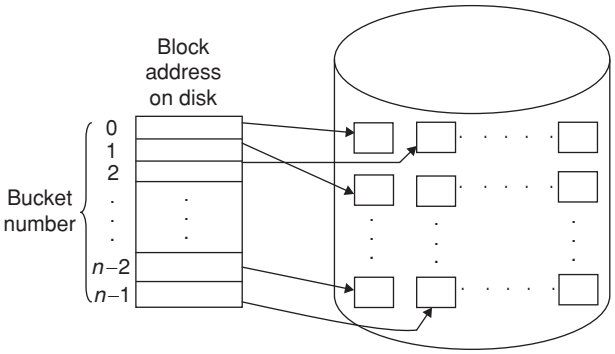
Multiple hashing The program applies a second hash function if the first results in a collision. If another collision results, the program uses open addressing or applies a third hash function and then uses open addressing if necessary.

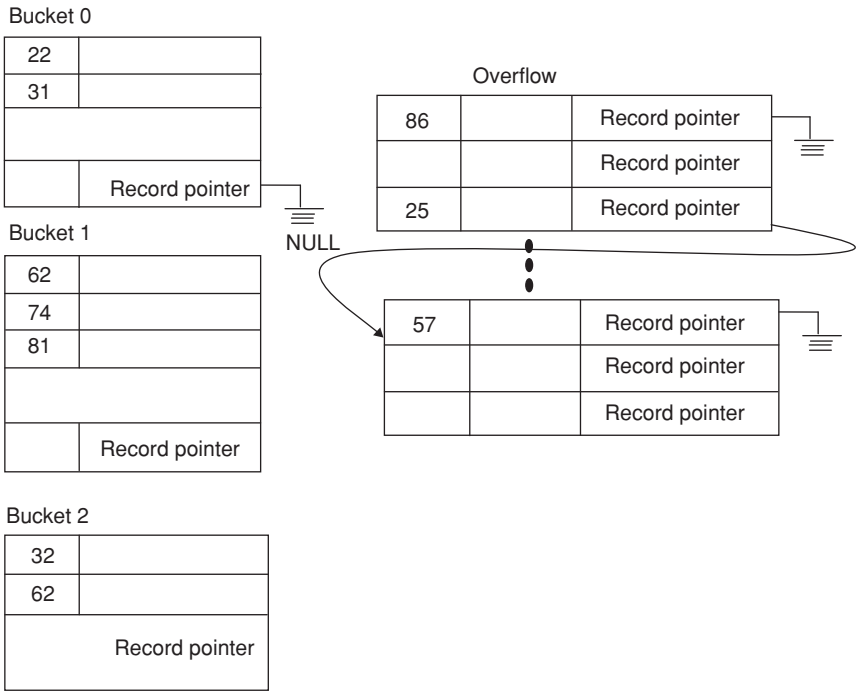
If we expect to have '*r*' records to store in the table, we should choose *M* locations for the address space such that (*r*/*M*) is between 0.7 and 0.9. It may also be useful to choose a prime number for *M*, since it has been demonstrated that this distributes the hash addresses better over the address space when the 'mod' hashing function is used. Other hash functions may require *M* to be a power of 2.

External Hashing

Hashing for disk files is called *external hashing*. To suit the characteristics of disk storage, the target address space is made of buckets, each of which holds multiple records. A bucket is either one disk block or a cluster of contiguous blocks. The hashing function maps a key into a relative bucket number, rather than assigning an absolute block address to the bucket. A table maintained in the file header converts the bucket number into the corresponding disk block address.

The collision problem is less severe with buckets, because as many records as will fit in a bucket can hash to the same bucket without causing problems. If the capacity of bucket exceeds, we can use a variation of chaining in which a pointer is maintained in each bucket to a linked list of overflow records for the bucket. The pointers in the linked list should be record pointers, which include both a block address and a relative record position within the block.





The hash function is $h(k) = k$ mod 10 and the hashing scheme described above is called *static hashing* because a fixed number of buckets *M* is allocated. It can be a drawback for dynamic files. Suppose that we allocate *M* buckets for the address space and let '*m*' be the maximum number of records that can fit in one bucket, then at most (*m* * *M*) records will fit in the allocated space. If the number of records turns out to be substantially fewer than (*m* * *M*), we are left with a lot of unused space.

If the number of records increases to substantially more than (*m* * *M*), numerous collisions will result and retrieval will be slowed down because of the long lists of overflow

records. In either case, we may have to change the number of blocks M allocated and then use a new hashing function (based on the new value of M) to redistribute the records. These organizations can be quite time consuming for large files. Newer dynamic file organizations based on hashing allows the number of buckets to vary dynamically with only localized reorganization.

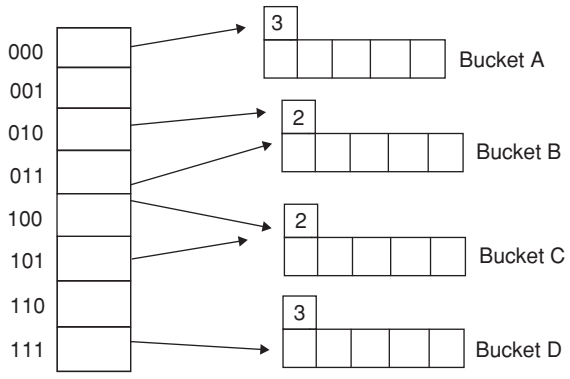
Hashing Techniques with Dynamic File Expansion

The disadvantage of static hashing is that the hash address space is fixed. Hence, it is difficult to expand or shrink the file dynamically. The first scheme is extendible hashing, It stores an access structure in addition to the file hence it is similar to indexing. The main difference is that the access structure is based on the values that result after application of the hash function to the search field. In indexing, the access structure based on the values of the search field itself. The second technique, called *linear hashing*, does not require additional access structure.

These hashing schemes take advantage of the fact that the result of applying a hashing function is a non-negative integer and hence can be represented as a binary number. The access structure is built on the binary representation of the hashing function result, which is a string of bits. We call this the *hash value of a record*. Records are distributed among buckets based on the values of the leading bits in their hash values.

Extendible Hashing

In extendible hashing, a type of directory, an array of 2^d bucket addresses is maintained, where d is called the *global depth of the directory*. The integer value corresponding to the first (high-order) d bits of a hash value is used as an index to the array to determine a directory entry, and the address in that determines the bucket in which the corresponding records are stored. Several directory locations with the same first d' bits for their hash values may contain the same bucket address if all the records that hash to these locations fit in a single bucket. A local depth d' is stored with each bucket specifies the number of bits on which the bucket contents are based.



The value of d can be increased and decreased by one at a time, thus doubling or halving the number of entries in the directory array. Doubling is needed if a bucket, whose local depth d' is equal to the global depth d, overflows. Halving occurs if $d > d'$ for all the buckets after some locations occur. Most record retrievals require two block accesses: one to the directory and the other to the bucket.

The main advantage of extendible hashing is the performance of the file does not degrade as the file grows, as opposed to static external hashing where collisions increases and the corresponding chaining causes additional accesses. No space is allowed in extendible hashing for future growth, but additional buckets can be allocated dynamically as needed. The space overhead for the directory table is negligible.

Another advantage is that splitting causes minor reorganization in most cases, since only the records in one bucket are redistributed to the two new buckets. The only time a reorganization is more expensive is when the directory has to be doubled (or) halved.

A disadvantage is that the directory must be searched before accessing the buckets themselves, resulting in two block accesses instead of one in static hashing.

Indexing

Indexes are auxiliary access structures, which are used to speed up the retrieval of records in response to certain search conditions. The index structure typically provides secondary access paths, which provide alternative ways of accessing the records without affecting the physical placement of records on disk. They enable efficient access to records based on the indexing fields that are used to construct the index.

Any field of the file can be used to create an index and multiple indexes on different fields can be constructed on the same file. To find a record or records in the file based on a certain selection criterion on an indexing field, one has to initially access the index, which points to one or more blocks in the file where the required records are located. The most prevalent types of indexes are based on ordered files (single-level indexes) and tree data structures (multilevel indexes, B^+ trees).

Dense Index files: Index record appears for every search-key value in the file.

Brighton	*A*-217	Brighton	750
Downtown	*A*-101	Downtown	600
Mianus	*A*-110	Downtown	300
Perryridge	*A*-215	Mianus	400
	*A*102	Perryridge	800

Figure 3 Dense index file.

Sparse index files These files contain index records for only some search-key values. Applicable when records are sequentially ordered on search key.

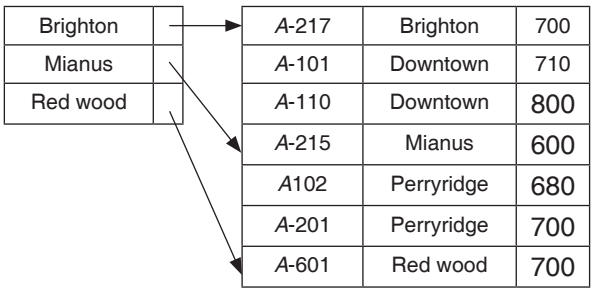


Figure 4 Sparse index file.

Compared to dense index, sparse index takes less space and less maintenance over head for insertions and deletions. It is slower than dense index for locating records.

Index Update

Record deletion If delete key was the only record in the file with its particular search-key value, the search key is deleted from the index also.

In dense index, delete the search key.

In spare index, if deleted key value exists in the index, the value is replaced by next search-key value in the file. If the next search-key value already has an index entry, the entry is deleted instead of being replaced.

Record insertion In dense index, if the search-key value doesn't appear in the index insert it.

If index stores an entry for each block of the file, no change needs to be made to the index unless a new block is created. If a new block is created, the first search-key value appearing in the new block is inserted into the index.

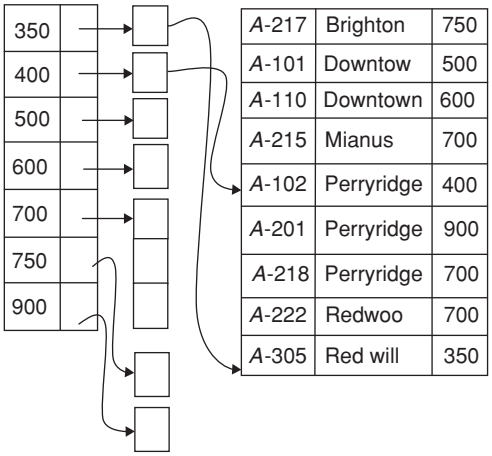


Figure 5 Secondary index.

Secondary index example:

1. Index record points to a bucket that contains pointers to all the actual records with that particular search – key value
2. secondary index have to be dense

Single-level Ordered Indexes

A file with a given record structure consisting of several fields (or attributes), an index access structure is usually defined on a single field of a file is called an *indexing field* or *indexing attribute*. The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value. The values in the index are ordered so that we can do a binary search on the index.

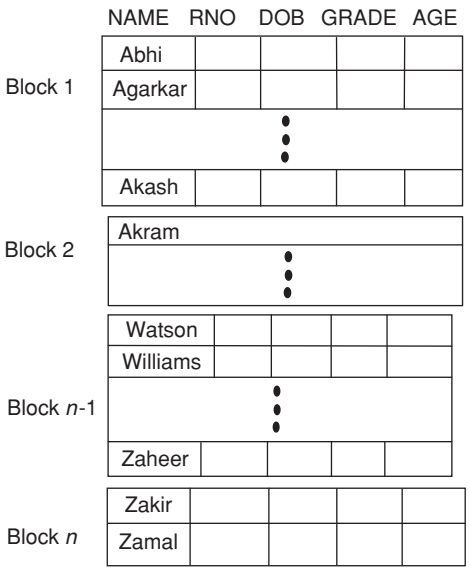
The index file is much smaller than the data file, so searching the index using a binary search is reasonably efficient. Multilevel indexing does away the need for a binary search at the expense of creating indexes to the index itself.

Types of Ordered Indexes

1. Primary index
2. Clustering index
3. Secondary index

Primary index A primary index is an ordered file whose records are of fixed length with two fields. The first field is of the same data type as the ordering key field called the *primary key of the data file*, and the second field is a pointer to a disk block (block address). There is one index entry (index record) in the index file for each block in the data file. Each index entry has the value of the primary key field for the record in a block and a pointer to that block as its two field values. The two field values of index entry i is $<k(i), p(i)>$.

Example:



To create a primary index on the ordered file shown in the above figure, we use the NAME field as primary key, because that the ordering key field on the file (assuming that each value of NAME is unique). Each entry in the index has a NAME value and a pointer. Some sample index entries are as follows:

< $k(1)$ = (Abhi), $p(1)$ = address of block 1 >
< $k(2)$ = (Akram), $p(2)$ = address of block 2 >
< $k(3)$ = (Brat), $p(3)$ = address of block 3 >

The below figure illustrates this primary index. The total number of entries in the index is the same as the number of disk blocks in the ordered data file. The first record in each block of the data file is called the *anchor record of the block (or) block anchor*.

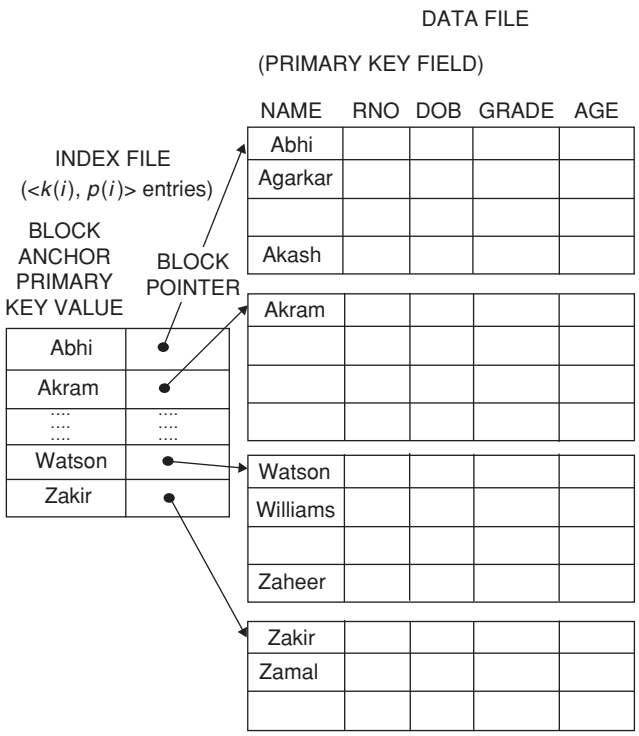


Figure 6 Primary index on the ordering key field of the file.

Indexes can also be characterized as dense or sparse. A dense index has an index entry for every search-key value (every record) in the data file. A sparse (non-dense) index has index entries only for some of the search values. A primary index is non-dense (sparse) index, since it includes an entry for each disk block of the data file and the keys of its anchor record rather than for every search value.

The index file for primary index needs fewer blocks than does the data file, for two reasons as follows:

1. There are fewer index entries than there are records in the data file.
2. Each index entry is typically smaller in size than a data record because it has only two fields. So more index entries than data records can fit in one block.

A binary search on the index file requires fewer block accesses than a binary search on the data file. The binary search for an ordered data file required $\log_2{}^b$ block accesses. But if the primary index file contains b_i blocks, then to locate a record with a search-key value requires a binary search of that index and access to the block containing that record, a total of $\log_2{}^b{}_i + 1$ accesses.

A record whose primary key value is k lies in the block whose address is $p(i)$, where $k(i) \le k \le k(i + 1)$. The ith block in the data file contains all such records because of the physical ordering of the file records on the primary key field. To retrieve a record, given the value k of its primary key field, we do a binary search on the index file to find the appropriate index entry i, and then retrieve the data file block whose address is $p(i)$.

The following example illustrates the saving in block accesses that is attainable when a primary index is used to search for a record.

Example: Suppose that we have an ordered file with r = 24,000 records stored on a disk with block size B = 512 bytes. File records are of fixed size and are unspanned, with record length R = 120 bytes.

The blocking factor for the file would be $bfr = \lfloor B/R \rfloor$

$$= \left\lfloor \frac{512}{120} \right\rfloor = \lfloor 4.26 \rfloor = 4 \text{ records per block}$$

The number of blocks needed for the file is

$$b = \left\lceil \left(\frac{r}{bfr} \right) \right\rceil = \left\lceil \frac{24,000}{42} \right\rceil = 6000 \text{ blocks}$$

A binary search on the data file would need

$\lceil \log_2{}^b = \log_2{}^{6000} \rceil = 13$ block accesses

Example: For the above data, suppose that the ordering key field of the file is V = 7 bytes long, a block pointer, P = 5 bytes long, and we have constructed a primary index for the file.

The size of each index entry is $R_i = (7 + 5) = 12$ bytes, so the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor$

$\lfloor 512/12 \rfloor = \lfloor 42.66 \rfloor = 42$ entries per block.

The total number of index entries r_i is equal to number of blocks in the data file, which is 6000. The number of index blocks is hence

$b_i = \lceil (r_i/bfr_i) \rceil = \lceil 6000/42 \rceil = 142$ blocks

To perform a binary search on the index file would need $\lceil \log_2 b_i \rceil = \lceil \log_2 142 \rceil = 8$ block accesses. To search for a record using the index, we need one additional block access to the data file for a total of '9' block accesses.

Disadvantage: A major problem with a primary index is insertion and deletion of records. If we attempt to insert a record in its correct position in the data file, we have to not only move records to make space for the new record but also change some index entries.

Clustering Index If records of a file are physically ordered on a non-key field, which does not have a distinct value for each record, that field is called the *clustering field*. We can create a different type of index called *clustering index* to

speed up the retrieval of records that have the same value for the clustering field.

A clustering index is also an ordered file with two fields, the first field is of the same type as the clustering field of the data file, and the second field is a block pointer.

There is one entry in the clustering index for each distinct value of the clustering field, containing the value and a pointer to the first block in the data file that has a record with that value for its clustering field.

The record insertion and deletion still cause problems, because the data records are physically ordered. To alleviate the problem of insertion, reserve a whole block (or a cluster of contiguous blocks) for each value of the clustering field, all records with that value are placed in the block (or block cluster). A clustering index is an example of a non-dense index, because it has an entry for every distinct value of the indexing field which is a non-key.

Secondary Index A secondary index provides a secondary means of accessing a file for which some primary access already exists. The secondary index may be on a field which is a candidate key and has a unique value in every record, or a non-key with duplicate values. The index is an ordered file with two fields. The second field is either a block pointer or a record pointer. There can be many secondary indexes for the same file.

First consider a secondary index access structure on a key field that has a distinct value for every record such a field is some times called a *secondary key*.

The records of the data file are not physically ordered by values of the secondary key field, we cannot use block anchors. That is why an index entry is created for each record in the data file, rather than for each block, as in the case of a primary index.

B⁻ Trees

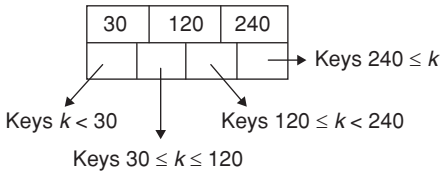
1. A commonly used index structure
2. Non-sequential, 'balanced'
3. Adapts well to insertions and deletions
4. Consists of blocks holding at most n keys and $n + 1$ pointers.
5. We consider a variation actually called a B^+ *tree*

B⁺ Trees

B^+ trees are a variant of B^- trees. In B^+ trees data stored only in leaves, leaves form a sorted linked list.

Parameter – n

Branching factor – $n + 1$

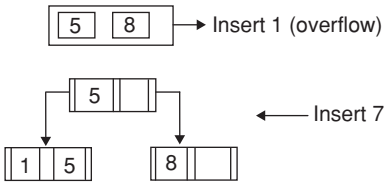


Each node (except root) has at least $n/2$ keys. B^- tree stands for balanced tree. All the paths through a B^- tree from root to different leaf nodes are of the same length (balanced path length). All leaf nodes are at the same depth level.

This ensures that number of disk accesses required for all the searches are same. The lesser the depth (level) of an index tree, the faster the search.

Insertion into B^+ tree

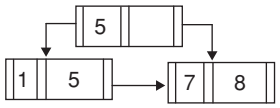
Given nodes 8 5 1 7 3 12 Initially start with root node (has no children)



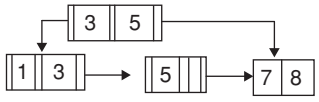
Overflow in Leaf Node

Split the leaf node First, j = ceiling $((p_{leaf} + 1)/2)$ entries are kept in the original node and the remaining moved to the new leaf.

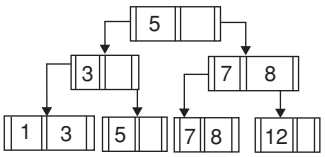
1. Create a new internal node, and jth index value is replicated in the parent internal node.
2. A pointer is added to the newly formed leaf node.



Insert 3 → overflow



Insert 12 (overflow, split propagates, new level)



Overflow in Internal Node

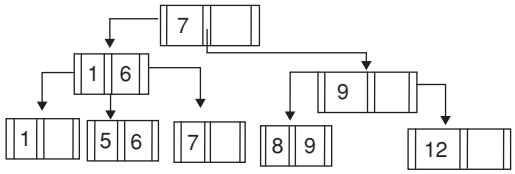
Split the internal node, the entries up to P_j where j = floor $((p + 1)/2)$ are kept in the original node and remaining moved to the new internal node

1. Create a new internal node and the jth index value is moved to the parent internal node (without replication)
2. Pointers are added to the newly formed nodes.

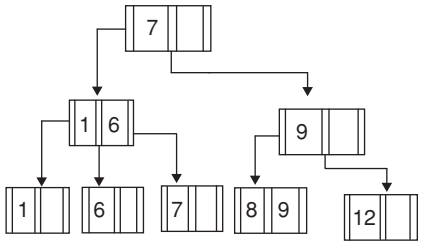
3. B^+ tree ensures some space always left in nodes for new entries. Also makes sure all nodes are at least half full.

Deletion in B^+ Trees

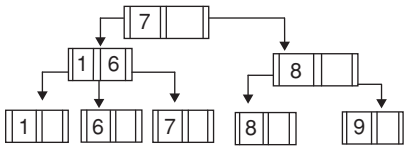
Delete 5,12,9 from the below B^+ tree:



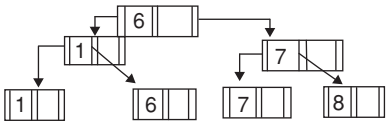
Delete 5:



Delete 12:
Under flow has occurred, so redistribute.



Delete 9: Underflow (merge with left) redistribute.



Advantages

1. B^- *Trees and* B^+ *trees*: B^- tree is a data structure used for external memory.
2. B^- trees are better than binary search trees if data is stored in external memory.
3. Each node in a tree should correspond to a block of data.
4. Each node can store many data items and has many successors.
5. The B^- tree has fewer levels but search for an item takes more comparisons at each level.
6. If a B^- tree has order '*d*', then each node (except root) has at least d/2 children, then the depth of the tree is at most $\log_{d/2}$ (size) + 1.
7. In the worst case, we need $(d - 1)$ comparisons in each node (using linear search)
8. Fewer disk accesses are required compared to binary Tree.
9. The usual data structure for an index is the B^+ tree.
10. Every modern DBMS contains some variant of B^- trees in addition with other index structures depending on the application.
11. B^- trees and B^+ trees are one and the same. They differ from B^- trees in having all data in the leaf blocks.
12. Compared to binary trees, B^- trees will have higher branching factor.
13. Binary trees can degenerate to a linear list, B^- trees are balanced, so this is not possible.
14. In B^+ tree, the values in inner nodes are repeated in the leaf nodes.
15. The height of the tree might decrease, because the data pointer is needed only in the leaf nodes, we can also get a sorted sequence.
16. In B^- trees, all leaves have the same distance from root hence B^- trees are balanced. This ensures that the chain of links followed to access a leaf node is never too long.
17. The time complexity of search operation in B^- tree (tree height) is $O(\log n)$, where 'n' is the number of entries.
18. Advantage of B^+ tree automatically reorganizes itself with small and local changes while doing insertions and deletions, reorganization of entire file is not required to maintain performance.
19. Disadvantage of B^+ tree, extra Insertion and deletion overhead, space overhead.
20. B^+ trees can be used as dynamic multilevel Indexes.

Exercises

Practice Problems I

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Consider the following specifications of a disk. Block size of a disk is 512 bytes, inter-block gap size is 128 bytes Number of blocks per track is 20 and number of tracks per surface is 400.

(i) What is the capacity of disk including Inter block gap?

(A) 124000 (B) 1260000
(C) 5120000 (D) 512000

(ii) What is the capacity of disk excluding Inter block gap?

(A) 25400 (B) 25600
(C) 25800 (D) 25900

2. Consider the following specifications of a disk. Block size of disk is 512 bytes, 2000 tracks per surface, 50 sectors per track and 5 double sided platters.

(i) What is the capacity of track in bytes?
(A) 4096000 (B) 4086000
(C) 4076000 (D) 4066000

(ii) What is the capacity of surface in bytes?
(A) 25600000 (B) 512000
(C) 5120000 (D) 51200000

(iii) What is the capacity of disk in bytes?
(A) 512*10^4 (B) 512*10^5
(C) 512*10^6 (D) 512*10^7

(iv) How many cylinders does it have?
(A) 512 (B) 1000
(C) 2000 (D) 2048

(v) Identify the In valid disk block size from below:
(A) 2048 (B) 51200
(C) 4098 (D) 4096

3. What is the order of internal node of B^+ tree suppose that a child pointer takes 6 bytes, the search field value takes 14 bytes and the block size is 512 bytes?
(A) 23 (B) 24
(C) 25 (D) 26

4. The order of a leaf node in a B^+ tree is the maximum number of (value, data, record pointer) pairs it can hold. Given that block size is 1 k bytes (1024 bytes), data record pointer is 7 bytes long, the value field is '9' bytes long and block pointer is 6 bytes.
(A) 63 (B) 64
(C) 65 (D) 66

5. The following key values are inserted into a B^+ tree in which order of the internal nodes is 3, and that of the leaf nodes is 2, in the sequence given below. The order of internal nodes is the maximum number of tree pointers in each node, and the order of leaf nodes is the maximum number of data items that can be stored in it. The B^+ tree is initially empty. 10, 3, 6, 8, 4, 2, 1. What is the maximum number of times leaf nodes would get split up as a result of these insertions?
(A) 3 (B) 4
(C) 5 (D) 6

6. For the same key values given in the above question, suppose the key values are inserted into a B^- tree in which order of the internal nodes is 3 and that of leaf nodes is 2. The order of internal nodes is the maximum number of tree pointers in each node and the order of leaf nodes is the maximum number of data items that can be stored in it. The B^- tree is initially empty. What is the maximum number of times leaf nodes would get split up as a result of these insertions?
(A) 1 (B) 2
(C) 3 (D) 4

7. Suppose that we have an ordered file with 45,000 records stored on a disk with block size 2048 bytes. File records are of fixed size and are unspanned with record length 120 bytes.

(i) What is the blocking factor?
(A) 16 (B) 17
(C) 18 (D) 19

(ii) What is the number of blocks needed for the file?
(A) 2642 (B) 2644
(C) 2646 (D) 2648

(iii) How many block accesses are required to search for a particular data file using binary search?
(A) 10 (B) 11
(C) 12 (D) 13

8. Suppose that the ordering key field of the file is 12 bytes long, a block pointer is 8 bytes long, and we have constructed a primary index for the file. Consider the file specifications given in the above questions.

(i) What is the size of each index entry?
(A) 16 (B) 18
(C) 20 (D) 22

(ii) What is the blocking factor for the index?
(A) 101 (B) 102
(C) 103 (D) 104

(iii) What is the total number of index entries?
(A) 2642 (B) 2644
(C) 2646 (D) 2648

(iv) What is the number of index blocks?
(A) 22 (B) 24
(C) 26 (D) 28

(v) How many block accesses are required, if binary search is used?
(A) 3 (B) 4
(C) 5 (D) 6

9. For the file specifications given in Q. No. 7, if we construct secondary index on a non-ordering key field of the file that is 12 bytes long, a block-pointer of size 8 bytes, each index entry is 20 bytes long and the blocking factor is 102 entries per block.

(i) What is the total number of index blocks?
(A) 422 (B) 424
(C) 442 (D) 444

(ii) How many block accesses are required to access the secondary index using binary search?
(A) 6 (B) 7
(C) 8 (D) 9

10. For the file specifications given in Q. No. 8, if we construct a multilevel index, number of 1st-level blocks are 442, blocking factor is 102, each index entry is 20 bytes long.

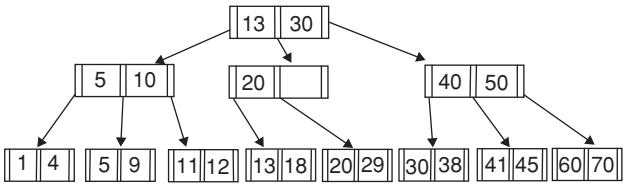
(i) What is the number of 2nd-level blocks?
(A) 4 (B) 5
(C) 6 (D) 7

(ii) What is the number of 3rd-level blocks?
(A) 0 (B) 1
(C) 2 (D) 3

11. Construct a B^+ tree for (1,4,7,10,17,21,31) with $n = 4$, which nodes will appear two times in a final B^+ tree?
(A) 17,7,20 (B) 17,7,20,25
(C) 17,20,25 (D) 7,17.25

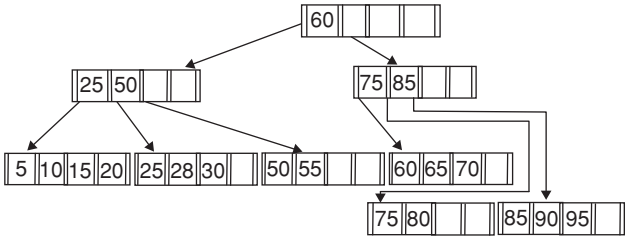
12. Suppose the hash function is $h(x) = x \bmod 8$ and each bucket can hold at most two records. The extendable hash structure after inserting 1, 4, 5 ,7, 8, 2, 20, what is the local depth of '4'?
(A) 0 (B) 1
(C) 2 (D) 3

13. Consider the given B^+ tree, insert 19 into the tree, what would be the new element in level 2?



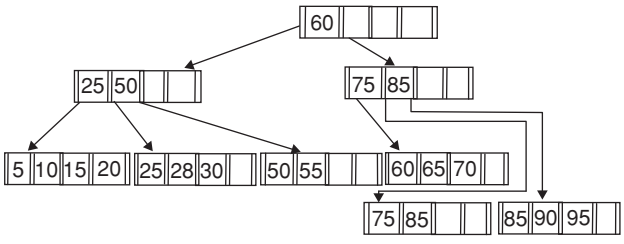
(A) 13 (B) 18
(C) 20 (D) 29

14. Consider the given B^+ tree, delete 70 and 25 from the tree, what are the elements present in level 2? ($\therefore$ root is at level 1)



(A) 25, 50, 75 (B) 25, 50,75, 85
(C) 28, 50, 75, 85 (D) 28, 50, 65, 75

15. Delete 60 from the above given tree (Q. No. 14). After deletion, what is the total number of nodes present in the tree?



(A) 5 (B) 6
(C) 7 (D) 8

16. What will be the number of index records/block?
(A) 68 (B) 65
(C) 69 (D) None

17. What will be the number of index blocks?
(A) 442 (B) 440
(C) 400 (D) None

18. Consider the following:
Block size = 1025 bytes
Record length in data file = 100 bytes
Total number of records = 30000
Search key = 9 bytes
Pointer = 6 bytes
What is the number of index blocks?
(A) 44 (B) 45
(C) 46 (D) None

19. Which of the following is maximum search time t_{max} in B^- trees?

(A) $t_{max} = a \log_2 \left(\frac{N}{2}\right)\left[\frac{a+d}{\log_2 m} + \frac{bm}{\log_2 m} + c\right]$

(B) $t_{max} = a \log_2 \left(\frac{N}{2}\right)\left[\frac{a+d}{\log_2 m} + \frac{bm}{\log_2 m} + c\right]$

(C) $t_{max} = a \log_2 N\left[\frac{a+d}{\log_2 m} + \frac{bm}{\log_2 m} + c\right]$

(D) $t_{max} = a \log_2 (N)\left[\frac{a}{\log_2 m} + \frac{bm}{\log_2 m} + c\right]$

20. Consider a B^+ tree. A child pointer takes 3 bytes, the search field value takes 7 bytes, and the block size is 256 bytes. What is the order of the internal node?
(A) 63 (B) 64
(C) 65 (D) 66

Practice Problems 2

Directions for questions 1 to 20: Select the correct alternative from the given choices.

1. Which of the following is true?
(A) Every conflict serializable is view serializable
(B) Every view serializable is conflict serializable
(C) Both A and B
(D) A schedule can be either only conflict serializable or only view-serializable.

2. Which one is the 2-phase locking rule?
(A) Two transactions cannot have conflicting locks
(B) No unlock operation can precede a lock operation in the same transaction.
(C) No data is/are affected until all locks are obtained and until the transaction is in its locked point.
(D) All of the above

3. If Transaction T_i has obtained an exclusive mode lock on item Q, then
(A) T_i can read Q
(B) T_i can write Q
(C) T_i can read and write
(D) Neither read nor write

4. Phantom phenomenon is
(A) A transaction retrieves a collection of objects but sees same result.
(B) A transaction retrieves a collection of objects but sees different results.
(C) Transaction T_1 waits for T_2 and T_2 waits for T_1
(D) This problem arises when the transaction has not locked all the objects.

5. We can avoid the starvation of transactions by granting locks by following manner:

When a transaction T_i requests a lock on a data item Q in a particular mode M, the concurrency control manager grants the lock provided that
(A) There is no other transaction holding a lock on Q in a mode that conflicts with M.
(B) There is no other transaction that is waiting for a lock on Q,
(C) (A) and (B)
(D) None

6. Which one is correct?
(A) Upgrading can take place only in shrinking phase
(B) Upgrading can take place only in growing phase.
(C) Downgrading can take place only in growing phase
(D) (A) and (C) both

7. A simple but widely used scheme automatically generates the appropriate lock and unlock instructions for a transaction, on the basis of read and write requests from the transaction:
(A) When a transaction T_i issues a read (Q) operation, the system issues a lock $s(Q)$ instruction followed by the read instruction.
(B) When T_i issues a write Q operation, the system checks to see whether T_i already holds a shared lock on Q. If it does, then the system issues an upgrade Q instruction followed by the write Q instruction, otherwise the system issues a lock -$X(Q)$ instruction, followed by the write Q instruction.
(C) All locks obtained by a transaction are unlocked after that transaction commits or aborts.
(D) All of the above

8. Which one is correct?
(A) A lock manager can be implemented as a process that receives messages from transactions and sends messages in reply.
(B) It uses linked list of records.
(C) It uses hash table called lock table.
(D) All of the above

Common data questions 9 and 10: Transaction T_1 has 5 instructions. Transaction T_2 has 3 instructions.

9. The number of non-serial transactions will be
(A) 15 (B) 8
(C) 2 (D) 56

10. The number of serial transaction schedules will be
(A) 15 (B) 8
(C) 2 (D) 56

11. In a heap file system, which of the following function finds 'average number of blocks to be read'?

(A) $\frac{i}{n} = \frac{1}{2}(1+n) = \frac{n}{2}$

(B) $\sum_{i=1}^{n} \frac{i}{n} = \frac{1}{2}(1+n) = \frac{n}{2}$

(C) $\sum_{i=0}^{n-1} \frac{i}{n} = \frac{1}{2}(1+n) = \frac{n}{2}$

(D) All of the above

12. What is the disadvantage in one directory per user?
(A) Different applications can be divided into separate groups.
(B) Different applications cannot be divided into separate groups
(C) All files are in a single group
(D) All of the above

13. What are the possible violations if an application program uses isolation-level 'Read uncommitted'?
(A) Dirty read problem
(B) Non-repeatable read problem
(C) Phantom phenomenon
(D) All of the above

14. The two-phase locking protocol
(A) ensures serializability
(B) issues locks in two phases
(C) unlocks in two phases
(D) All of the above

15. The point in the schedule where the transaction has obtained its final lock (the end of its growing phase) is called the
(A) block point (B) critical section
(C) growing point (D) lock point

16. Which of the following is not a problem of file management system?
(A) Data redundancy
(B) Lack of data independence
(C) Program dependence
(D) All of the above

17. Which of the following is/are true about master list of an index file?
(i) Is sorted in ascending order
(ii) A number is assigned to each record.
(A) Only (i) (B) Only (ii)
(C) Both (i) and (ii) (D) None of the above

18. To have a file, holding a list is necessary to
(i) Identify the records in the list
(ii) Identify the name, and type of the fields of each record.
(iii) Decide which fields will be used as sort of index keys.
(A) Only (i) and (ii)
(B) Only (i) and (iii)
(C) Only (ii) and (iii)
(D) All of the above

19. Two files may be joined into a third file, if the following is true:
(A) if they have row in common
(B) if they have a field in common
(C) Both (A) and (B)
(D) None

20. The minimum number of record movements required to merge four files *w*(with 10 records), *x*(with 20 records), *y*(with 15 records) and *z*(with 5 records) is:
(A) 50 (B) 40
(C) 30 (D) 35

Previous Years' Questions

1. A clustering index is defined on the fields which are of type **[2008]**
(A) non-key and ordering
(B) non-key and non-ordering
(C) key and ordering
(D) key and non-ordering

2. A B^- tree of order 4 is built from scratch by 10 successive insertions. What is the maximum number of node splitting operations that may take place? **[2008]**
(A) 3 (B) 4
(C) 5 (D) 6

3. Consider a file of 16384 records. Each record is 32 bytes long and its key field is of size 6 bytes. The file is ordered on a non-key field, and the file organization is unspanned. The file is stored in a file system with block size 1024 bytes, and the size of a block pointer is 10 bytes. If the secondary index is built on the key field of the file, and a multilevel index scheme is used to store the secondary index, the number of first-level and second-level blocks in the multilevel index are respectively **[2008]**
(A) 8 and 0 (B) 128 and 6
(C) 256 and 4 (D) 512 and 5

4. The following key values are inserted into a B^+ tree in which order of the internal node s is 3, and that of the leaf nodes is 2, in the sequence given below. The order of internal nodes is the maximum number of tree pointers in each node, and the order of leaf nodes is the maximum number of data items that can be stored in it. The B^+ tree is initially empty.

10, 3, 6, 8, 4, 2, 1

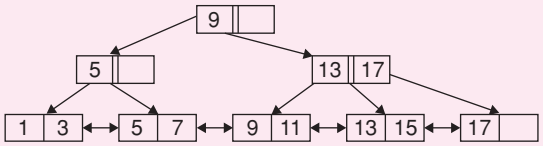
The maximum number of times leaf nodes would get split up as a result of these insertions is **[2009]**
(A) 2 (B) 3
(C) 4 (D) 5

5. Consider a B^+ tree in which the maximum number of keys in a node is 5. What is the minimum number of keys in any non-root node? **[2010]**
(A) 1 (B) 2
(C) 3 (D) 4

6. An index is clustered, if **[2013]**
(A) it is on a set of fields that form a candidate key.
(B) it is on a set of fields that include the primary key.
(C) the data records of the file are organized in the same order as the data entries of the index.
(D) the data records of the file are organized not in the same order as the data entries of the index.

7. A file is organized so that the ordering of data records is the same as or close to the ordering of data entries in some index. Then that index is called **[2015]**
(A) Dense (B) Sparse
(C) Clustered (D) Unclustered

8. With reference to the B+ tree index of order 1 shown below, the minimum number of nodes (including the Root node) that must be fetched in order to satisfy the following query: "Get all records with a search key greater than or equal to 7 and less than 15" is _____ **[2015]**



9. Consider a B^+ tree in which the search key is 12 bytes long, block size is 1024 bytes, record pointer is 10 bytes long and block pointer is 8 bytes long. The maximum number of keys that can be accommodated in each non-leaf node of the tree is _____. **[2015]**

10. B+ Trees are considered **BALANCED** because **[2016]**

(A) The lengths of the paths from the root to all leaf nodes are all equal.

(B) The lengths of the paths from the root to all leaf nodes differ from each other by at most 1.

(C) The number of children of any two non - leaf sibling nodes differ by at most 1.

(D) The number of records in any two leaf nodes differ by at most 1.

11. In a B+ tree, if the search-key value is 8 bytes long, the block size is 512 bytes and the block pointer size is 2 bytes, then the maximum order of the B+ tree is _____. **[2017]**

Answer Keys

Exercises

Practice Problems 1

1. (i) C (ii) A **2.** (i) B (ii) D (iii) C (iv) C (v) C **3.** C **4.** A **5.** C **6.** B
7. (i) B (ii) D (iii) C **8.** (i) C (ii) B (iii) D (iv) C (v) C **9.** (i) C (ii) D
10. (i) B (ii) B **11.** B **12.** D **13.** B **14.** C **15.** B **16.** A **17.** A **18.** B
19. A **20.** C

Practice Problems 2

1. A **2.** D **3.** C **4.** B **5.** C **6.** B **7.** D **8.** D **9.** D **10.** C
11. B **12.** B **13.** D **14.** D **15.** D **16.** D **17.** B **18.** D **19.** B **20.** B

Previous Years' Questions

1. A **2.** C **3.** C **4.** C **5.** B **6.** C **7.** C **8.** 5 **9.** 50 **10.** A
10. 52

Test

Databases

Time: 60 min.

1. Which of the following are used in DBMS files?
(i) Data dictionary (ii) DML
(iii) Query language (iv) Transaction log
(A) (i) and (ii) (B) (ii) and (iii)
(C) (iii) and (iv) (D) (i) and (iv)

2. Which among the following is not a problem of file management system?
(A) Data redundancy
(B) Lack of data independence
(C) Program independence
(D) None of these

3. A transparent DBMS
(A) cannot hide sensitive information from users
(B) keeps its logical structure hidden from users
(C) keeps its physical structure hidden from users
(D) All of the above

4. If the field size is too small, for the longest piece of data to be entered,
(A) database program will be freezed
(B) field will automatically expand
(C) part of the data will be cut off
(D) All of the above

5. Which of the following functional dependencies are satisfied by the instance from the below relation?

A	*B*	*C*
1	7	3
1	9	5
1	11	5
5	3	3

(A) $AB \rightarrow C$ and $C \rightarrow B$
(B) $BC \rightarrow A$ and $B \rightarrow C$
(C) $BC \rightarrow A$ and $A \rightarrow C$
(D) $AC \rightarrow B$ and $B \rightarrow A$

6. Let E_1 and E_2 be two entities in an E/R diagram with single-valued attributes, R_1 and R_2 are two relationships between E_1 and E_2, R_1 is one to many R_2 is many-to-one. R_1 and R_2 do not have any attributes of their own. What is the minimum number of tables required to represent this situation in the relation model?
(A) 2 (B) 3
(C) 4 (D) 5

7. Which of the following is true about DBMS?
(i) Low-level DMLs are record-at-a time
(ii) High-level DMLs are set oriented or set-at-a time
(iii) Query in high-level DML specify which data to retrieve rather than how.
(iv) When used as standalone, DML is called 'host language'
(A) (i) only (B) (i) and (iii)
(C) (i), (ii) and (iii) (D) (iii) and (iv)

8. In which of the following, the structure of data files is stored?
(A) Metadata (B) Database catalog
(C) Database schema (D) Data model

9. A schedule is a collection of
(A) Data models (B) Transactions
(C) Schemas (D) Tables

10. Select from the following which matches the term 'Impedance mismatch problem':
(A) In compatibility of storage and data structure
(B) Mismatch in user authentication
(C) File structure mismatching
(D) None of these

11. Which of the following is not a/an integrity constraint?
(A) Entity integrity
(B) Candidate key constraint
(C) Business rules
(D) None of the above

12. Select from the following which is concerned with 'Query Optimizer':
(A) Extracts DML commands from an application program in a high-level language
(B) Parsing and analyzing interactive query
(C) Rearrangement and reordering of operations and elimination of redundancies
(D) Performance monitoring

13. Which of the following does not belong to database model?
(A) Relational Model (B) Distributed Model
(C) Hierarchical Model (D) Network Model

14. What is the correct sequence of database design process?
(i) Create conceptual schema
(ii) Data model mapping
(iii) Requirement collection and analysis
(iv) Physical design
(A) iii →i →ii → iv
(B) iii → ii → i → iv
(C) i → ii → iii → iv
(D) i → iii → ii → iv

15. Consider the following schema definitions
Employee {Name, SSN, Address, DNo}
Department {DName, DNumber, Manager, SSN}
Which among the following expressions represent the query $\Pi_{\text{name, address}}(\sigma_{\text{Dname = 'Res'} \wedge \text{ DNumber = DNo}}$ (Department $\bowtie$ Employee)?

(A) Retrieve the name and address of employees who work for the project no 'Dno'
(B) Retrieve the name and address of all employees who control the 'Res' department.
(C) Retrieve the name and address of all employees who work for the 'Res' department.
(D) None of these

16. Select from the following which closely resembles the concept 'Degree of a relationship':
(A) Number of entities participating in a relation
(B) Number of entity types participating in a relation
(C) Number of strong entity types in a relation
(D) Number of weak entity types in a relation

17. Consider the following statements in a database:
(i) No primary key value can be NULL
(ii) A tuple in one relation which refers to another relation must refer to an existing tuple in that relation
(iii) The value of x determines the value of y in all states of a relation, where x and y are two attributes of the relation Which of the following combinations matches the given statements in order?
(A) Referential integrity, functional dependency, entity integrity.
(B) Functional dependency, entity integrity, referential integrity
(C) Entity integrity, functional dependency, referential integrity.
(D) Entity integrity, referential integrity, functional dependency

18. Consider the following relation schemas:

Works (emp_name, comp_name,salary)

Livesin (emp_name, street, city)

Location (comp_name, city)

Manager (manager_name)

What is returned by the following relational algebra expression

$\pi_{\text{emp\_name}}(\sigma_{\text{comp-name=Time} \wedge \text{Works.emp\_name=livesin.emp\_name}})$

(Works $\bowtie$ Livesin)
(A) Names of all employees who work for TIME
(B) Names of all employees of TIME who lives in the same city
(C) Names of people who live in the same city
(D) None of these

19. Consider the following SQL query:
Select distinct $a_1, a_2, \ldots, a_n$ from $r_1, r_2 \ldots r_m$ where P
This query is equivalent to one of the following relational algebra expression:
(A) $\pi_{a_1, a_2 \ldots a_n}\sigma_P(r_1 \times r_2 \times \cdots \times r_m)$
(B) $\pi_{a_1, a_2 \ldots a_n}\sigma_P(r_1 r_2 \times \cdots \times r_m)$
(C) $\pi_{a_1, a_2 \ldots a_n}\sigma_P(r_1 \cup r_2 \cup \times \ldots \times \cup r_m)$
(D) $\pi_{a_1, a_2 \ldots a_n}\sigma_P(r_1 r_2 \times \ldots \times r_m)$

20. Let $R_1(A, B, C)$ and $R_2(D, E)$ be two relation schemas with primary keys A and D and C be a foreign key in R_1 referring to R_2. Suppose there is no violation of the above referential integrity constraint in the instances r_1 and r_2, which of the following relational algebra expression would necessarily produce an empty relation?
(A) $\pi_D(r_2) - \pi_C(r_1)$
(B) $\pi_C(r_1) - \pi_E(r_2)$
(C) $\pi_D(r_1 \bowtie_{C=D} r_2) - \pi_B(r_1)$
(D) $\pi_C(r_1 \bowtie_{C=E} r_2)$

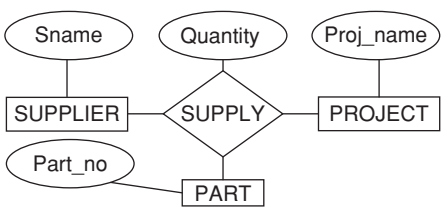
21. Let r be an instance for the schema $R = (A, B, C, D)$. Let $r_1 = \pi_{A, B, C}(r) r_2 = \pi_{A, D}(r)$ and $S = r_1 \bowtie r_2$. Also given that the decomposition of r into r_1 and r_2 is lossy, which of the following is true?
(A) $S \subset r$ (B) $r \cup s = r$
(C) $r \subset s$ (D) $r \bowtie s = s$

22. Which of the following is/are logical database structures?
(A) Network (B) Tree
(C) Chain (D) All of the above

23. A relational database management system manages data in more than one file at a time by using which of the following combinations?
(A) Tables and tuples
(B) Relations and tuples
(C) Tables and Relations
(D) Attributes and tuples

24. Let Emp = (Name, ID, ADDRESS, PHONE, SPOUSE, LIVINGAT) be a relation scheme with following FDs, which one of the following is a key
ADDRESS → Phone
SPOUSE → NAME
SPOUSE, ADDRESS → PHONE
NAME → ID
(A) ADDRESS, PHONE
(B) SPOUSE, ADDRESS
(C) NAME, SPOUSE
(D) NAME, ADDRESS

25. Consider the following E-R diagram



Select the most appropriate statement from the following for the above ER diagram:

(A) Represents a ternary relationship
(B) Represents a binary relationship
(C) Represents a ternary relationship with instances of the form (s, j, p)
(D) Represents 1 – to – many relationships

26. If two relations R_1 and R_2 are such that they are of the same degree and domain of the corresponding fields are also the same, then which one of the following is true about R_1 and R_2?
(A) $R_1 \subset R_2$
(B) $R_1 \cup R_2 = R_2 \cup R_1$
(C) R_1 and R_2 are union compatible
(D) None of these

Common data questions for 27 and 28: Let Employee and Guests be two relations with attributes (id, mobl_no, name, address) and (id, mob–no, comps_working, shifts) Relations respectively {id, mob_no} is the key for both.

27. Which of the following queries are equivalent?
(i) π_{id} (Employee $\bowtie$ Guests)
(ii) π_{id} (Employee) $\bowtie \pi_{id}$(Guests)
(iii) π_{id} {(Employee–Guest) $\cap$ Guest–Employee)}
(iv) π_{id} { $\pi_{id, mob}$(Employee) $\cap$ $\pi_{id, mob}$(Guest)}
(A) (ii) and (iii) (B) (ii), (iii) and (iv)
(C) (i), (ii) and (iv) (D) (ii) and (iv) only

28. What does the following relational algebra expression represent?
π_{id} (π_{id, mob_no}(Employee–Guests))
(A) Id of all employees working with the company
(B) Id of all permanent employees
(C) Id of part time employees
(D) None of these

Common data for questions 29 and 30:

29. Let R_1 and R_2 be two relations with attributes a_1 and a_2. P_1 and P_2 be two predicates.
Select the expression from the following which is wrong:
(A) $\sigma_{P_1}(\sigma_{P_1}(R_1)) \rightarrow \sigma_{P_2}(\sigma_{P_2}(R_1))$
(B) $\sigma_{P_1}(\pi_{a_1}(R_1)) \rightarrow \pi(\sigma_{a_1}(\sigma_{P_1}(R_1)))$
(C) $\sigma_{P_1}(R_1 \cup R_2) \rightarrow \sigma_{P_1}(R_1) \cup \sigma_{P_2}(R_2)$
(D) $\pi_{a_2}(\sigma_{a_1}(R_1)) \rightarrow \sigma_{P_1}(\pi_{a_2}(R_1))$

30. Select from the following corresponding TRC for the wrong expression in the above question:
(A) $\{t/\ \exists u, R_1(t[P_1]) = R_2(u[P_1])\}$
(B) $\{t/\ \forall u, R_1(t[P_1]) = R_1(u[P_1])\}$
(C) $\{t/\ \exists u, R_1(t[P_1]) \neq R_2(u([P_1])\}$
(D) $\{t/\square(t \in R_1)\}$

Answer Keys

1. D	**2.** D	**3.** C	**4.** C	**5.** B	**6.** B	**7.** C	**8.** B	**9.** B	**10.** A
11. B	**12.** C	**13.** B	**14.** A	**15.** C	**16.** A	**17.** D	**18.** C	**19.** A	**20.** A
21. C	**22.** D	**23.** C	**24.** B	**25.** C	**26.** C	**27.** C	**28.** B	**29.** A	**30.** B