Biyani's Think Tank

Concept based notes

Database Management System

(MCA)

Nandini Parwal
Information Technology
Biyani Institute of Science and Management
Jaipur

Published by :
Think Tanks
Biyani Group of Colleges

Concept & Copyright :
©Biyani Shikshan Samiti
Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 • Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

ISBN :

Edition : 2011
Price :

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self-explanatory and adopts the “Teach Yourself” style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

This book covers basic concepts related to the microbial understandings about diversity, structure, economic aspects, bacterial and viral reproduction etc.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this Endeavour. They played an active role in coordinating the various stages of this Endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

Syllabus

Overview of DBMS, Basic DBMS terminology, data base system v/s file system, data independence. Architecture of a DBMS

Introduction to data models: entity relationship model, hierarchical model: from network to hierarchical, relational model, comparison of network, hierarchical and relational models.

Data modeling using the Entity Relationship Model: ER model concepts, notation for ER diagram, mapping constraints, keys, Concepts of Super Key, candidate key, primary key, Generalization, aggregation, reduction of an ER diagrams to tables, extended ER model, relationships of higher degree.

Relational model: storage organizations for relations, relational algebra, relational calculus.

Normalization: Functional dependencies, normal forms, first, second, third normal forms, BCNF, inclusion dependencies, loss less join decompositions, normalization using FD, MVD, and JDs, alternative approaches to database design.

Introduction to SQL: Characteristics of SQL, Advantages of SQL, SQL data types and literals, Types of SQL commands, SQL operators and their procedure, Tables, views and indexes, Queries and sub queries, Aggregate functions, insert, update and delete operations, Joins, Unions, Intersection, Minus in SQL.

Contents

6.	**Relational Model and DBA** 6.1 Relational Algebra 6.2 Relational Calculus 6.3 DBA
7.	**Data and Query Processing** 7.1 Basic Retrieval Capacity 7.2 Query Language 7.3 Query Processing 7.4 Client/Server Design
8.	**Structured Query Languages** 8.1 Basic SQL & Keys 8.2 Nested Queries 8.3 QBEL & Quel 8.4 Aggregate Operators
9.	**Advanced Features of SQL** 9.1 Embedded SQL 9.2 Dynamic SQL 9.3 Cursors

Chapter 1

Overview of DBMS

Q.1 What do you mean by Data and Information?

Ans.: Data are plain facts. The word "data" is plural for "datum." When data are processed, organized, structured or presented in a given context so as to make them useful, they are called Information. It is not enough to have data (such as statistics on the economy). Data themselves are fairly useless, but when these data are interpreted and processed to determine its true meaning, they becomes useful and can be called Information.

Q.2 What do you mean by Database?

Ans.: Definitions of **Database** :

- An organized body of related information.
- In computing, a database can be defined as a structured collection of records or data that is stored in a computer so that a program can consult it to answer queries. The records retrieved in answer to queries become information that can be used to make decisions.
- An organized collection of records presented in a standardized format searched by computers. Web Pals, ID Weeks Library's Online Catalog, is a database. The periodical indexes available through the library are also databases.
- A collection of data organized for rapid search and retrieval by a computer.

- A collection of related data stored in one or more computerized files in a manner that can be accessed by users or computer programs via a database management system.
- An organized collection of information, data, or citations stored in electronic format that can be searched for specific information or records by techniques specific to each database.
- A logical collection of interrelated information, managed and stored as a unit, usually on some form of mass-storage system such as magnetic tape or disk.
- A database is a structured format for organizing and maintaining information that can be easily retrieved. A simple example of a database is a table or a spreadsheet.
- A database in an organized collection of computer records. The most common type of database consists of records describing articles in periodicals otherwise known as a periodical index.
- A database collects information into an electronic file, for example a list of customer addresses and associated orders. Each item is usually called a 'record' and the items can be sorted and accessed in many different ways.
- A set of related files that is created and managed by a Database Management System (DBMS).
- A computerized collection of information.
- Integrated data files organized and stored electronically in a uniform file structure that allows data elements to be manipulated, correlated, or extracted to satisfy diverse analytical and reporting needs.
- A collection of information stored in one central location. Many times, this is the source from which information is pulled to display products or information dynamically on a website.
- Relational data structure used to store, query, and retrieve information.

- An organized set of data or collection of files that can be used for a specified purpose. A collection of interrelated data stored so that it may be accessed with user friendly dialogs.
- A large amount of information stored in a computer system.

Q.3 What are the basic objectives of the Database?

Ans.: A database is a collection of interrelated data stored with minimum redundancy to serve many users quickly and efficiently. The general objective is to make information access easy. Quick, inexpensive, and flexible for the user. In data base design, several **specific objectives** are considered.

(i) Controlled Redundancy

(ii) Ease of Learning and Use

(iii) Data Independence

(iv) Most Information in Low Cost

(v) Accuracy and Integrity

(vi) Recovery from failure

(vii) Privacy and Security

(viii) Performance

Q.4 Name the elements of Database.

Ans.: Elements of Database as follows:

(i) Data Collection

(ii) Direct Access Storage Device

(iii) Data Dictionary

(iv) Database Administrator

(v) Database Management System

Q.5 Define the Database Management System.

Ans.: (i) A **Database Management System** (DBMS), or simply a **Database System** (DBS), consists of :

- A collection of interrelated and persistent data (usually referred to as the **Database** (DB)).
- A set of application programs used to access, update and manage that data (which form the data Management System (MS)).

(ii) The goal of a DBMS is to provide an environment that is both **convenient** and **efficient** to use in :

- Retrieving information from the database.
- Storing information into the database.

(iii) Databases are usually designed to manage **large** bodies of information. This involves :

- Definition of structures for information storage (data modeling).
- Provision of mechanisms for the manipulation of information (file and systems structure, query processing).
- Providing for the safety of information in the database (crash recovery and security).
- Concurrency control if the system is shared by users.

Q.6 Why do we need Database Management System? Describe in the term of development.

Ans.: There are four basic components of Database Management System :

(i) **Data :** Raw facts which we want to feed in the computer.

(ii) **Hardware :** On which the data to be processed.

(iii) **Software :** The interface between the hardware and user, by which the data will change into the information.

(iv) **User :** There are so many types of users some of them are application programmer, endcase users and DBA.

Q.7 What is the basic purpose of Database Management System?

Ans.: Purpose of Database Systems :

(i) To see why database management systems are necessary, let's look at a typical ``File-Processing System'' supported by a conventional operating system.

The application is a savings bank :

- Savings account and customer records are kept in permanent system files.
- Application programs are written to manipulate files to perform the following **tasks** :
  - Debit or credit an account.
  - Add a new account.
  - Find an account balance.
  - Generate monthly statements.

(ii) Development of the System proceeds as follows :

- New application programs must be written as the need arises.
- New permanent files are created as required.
- but over a long period of time files may be in different formats, and
- Application programs may be in different languages.

(iii) So we can see there are problems with the Straight File-Processing Approach :

- **Data Redundancy and Inconsistency :**
  - Same information may be duplicated in several places.
  - All copies may not be updated properly.

- **Difficulty in Accessing Data :**
  - May have to write a new application program to satisfy an unusual request.
  - E.g. find all customers with the same postal code.
  - Could generate this data manually, but a long job.
- **Data Isolation :**
  - Data in different files.
  - Data in different formats.
  - Difficult to write new application programs.
- **Multiple Users :**
  - Want concurrency for faster response time.
  - Need protection for concurrent updates.
  - E.g. two customers withdrawing funds from the same account at the same time - account has $500 in it, and they withdraw $100 and $50. The result could be $350, $400 or $450 if no protection.
- **Security Problems :**
  - Every user of the system should be able to access only the data they are permitted to see.
  - E.g. payroll people only handle employee records, and cannot see customer accounts; tellers only access account data and cannot see payroll data.
  - Difficult to enforce this with application programs.
- **Integrity Problems :**
  - Data may be required to satisfy constraints.
  - E.g. no account balance below $25.00.
  - Again, difficult to enforce or to change constraints with the file-processing approach.

These problems and others led to the development of **Database Management Systems**.

Q.8 What are the advantages and disadvantages of DBMS over Conventional File System?

Ans.: Advantages :

- An organized and comprehensiveness of recording the result of the firms activities.
- A receiver of data to be used in meeting the information requirement of the MIS users.
- Reduced data redundancy.
- Reduced updating errors and increased consistency.
- Greater data integrity and independence from applications programs.
- Improved data access to users through use of host and query languages.
- Improved data security.
- Reduced data entry, storage, and retrieval costs.
- Facilitated development of new applications program.
- Standard can be enforced: Standardized stored data format is particularly desirable as an old data to interchange or migration (change) between the system.
- Conflicting requirement can be handled.

Disadvantages :

- It increases opportunity for person or groups outside the organization to gain access to information about the firms operation.
- It increases opportunity for fully training person within the organization to misuse the data resources intentionally.
- The data approach is a costly due to higher H/W and S/W requirements.
- Database systems are complex (due to data independence), difficult, and time-consuming to design.
- It is not maintain for all organizations .It is only efficient for particularly large organizations.
- Damage to database affects virtually all applications programs.

- Extensive conversion costs in moving form a file-based system to a database system.
- Initial training required for all programmers and users.

Q.9 Why would you choose a database system instead of simply storing data in operating system files? When would it make sense not to use a database system?

Ans: A database is an integrated collection of data, usually so large that ithas to be stored on secondary storage devices such as disks or tapes. This data can be maintained as a collection of operating system files, or stored in a DBMS (database management system). The advantages of using a DBMS are:

Data independence and efficient access. Database application programs are independent of the details of data representation and storage. The conceptual and external schemas provide independence from physical storage decisions and logical design decisions respectively. In addition, a DBMS provides efficient storage and retrieval mechanisms, including support for very large files, index structures and query optimization.

Reduced application development time. Since the DBMS provides several important functions required by applications, such as concurrency control and crash recovery, high level query facilities, etc., only application-specific code needs to be written. Even this is facilitated by suites of application development tools available from vendors for many database management systems.

Data integrity and security. The view mechanism and the authorization facilitiesof a DBMS provide a powerful access control mechanism. Further, updates to the data that violate the semantics of the data can be detected and rejected by the DBMS if users specify the appropriate integrity constraints.

Data administration. By providing a common umbrella for a large collection ofdata that is shared by several users, a DBMS facilitates maintenance and data
administration tasks. A good DBA can effectively shield end-users from the chores of fine-tuning the data representation, periodic back-ups etc.

Concurrent access and crash recovery. A DBMS supports the notion of a transaction,which is conceptually a single user's sequential program. Users can writetransactions as if their programs were running in isolation against the database.

The DBMS executes the actions of transactions in an interleaved fashion to obtain good performance, but schedules them in such a way as to ensure that conflicting operations are not permitted to proceed concurrently. Further, the DBMS maintains a continuous log of the changes to the data, and if there is a system crash, it can restore the database to a transaction-consistent state. That is, the actions of incomplete transactions are undone, so that the database state reflects only the actions of completed transactions. Thus, if each complete transaction, executing alone, maintains the consistency criteria, then the database state after recovery from a crash is consistent.

If these advantages are not important for the application at hand, using a collection of files may be a better solution because of the increased cost and overhead of purchasing and maintaining a DBMS.

Q.10 What do you mean by flat file database?
Ans: It is a database in which there are no programs or user access languages. It has no cross-file capabilities but is user-friendly and provides user-interface management.

Q.11 What is "transparent DBMS"?
Ans: It is one, which keeps its Physical Structure hidden from user.

Q.12 What is durability in DBMS?
Ans: Once the DBMS informs the user that a transaction has successfully completed, its effects should persist even if the system crashes before all its changes are reflected on disk. This property is called durability.

Q.13 What is Data Independence?
Ans: Data independence means that "the application is independent of the storage structure and access strategy of data". In other words, The ability to modify

the schema definition in one level should not affect the schema definition in the next higher level.

Two types of Data Independence:

1. **Physical Data Independence:** Modification in physical level should not affect the logical level.
2. **Logical Data Independence:** Modification in logical level should affect the view level.

NOTE: Logical Data Independence is more difficult to achieve

Q.14 Explain the difference between logical and physical data independence.

Ans: Logical data independence means that users are shielded from changes in the logical structure of the data, while physical data independence insulates users from changes in the physical storage of the data. We saw an example of logical data independence in the answer to Exercise 1.2. Consider the Students relation from that example (and now assume that it is not replaced by the two smaller relations). We could choose to store Students tuples in a heap file, with a clustered index on the sname field. Alternatively, we could choose to store it with an index on the gpa field, or to create indexes on both fields, or to store it as a file sorted by gpa. These storage alternatives are not visible to users, except in terms of improved performance, since they simply see a relation as a set of tuples. This is what is meant by physical data independence.

Q.15 Does the relational model, as seen by an SQL query writer, provide physical and logical data independence? Explain.

Ans: The user of SQL has no idea how the data is physically represented in the machine. He or she relies entirely on the relation abstraction for querying. Physical data independence is therefore assured. Since a user can define views, logical data independence can also be achieved by using view definitions to hide changes in the conceptual schema.

Chapter 2

Database Normalization

Q.2.1 Explain Keys?

Ans 2.1 KEYS

A key is that data item that exclusively identifies a record. For example, Account number, Product code, Employee number and Customer number are used as key fields because they specifically identify a record stored in a database.

2.1.1 Super Key

A super Key for an entity is a set of one or more attributes whose combined value uniquely indentifies the entity in the entity set. For example, for an entity set Employees, the set of attributes (emp_name, address) can be considered to be a super key, if we assume that there are no two employees with the same emp_name as well as the same address.

2.1.2 Primary Key

The primary key uniquely identifies each record in a table and must never be the same for two records. For example, emp_code can be primary key for the entity set Employees.

The primary key should be chosen such that its attributes are never or very rarely changed. For instance, the address field of a person should not be part of the primary key, since it is likely to change. Emp_code, on the other hand, is not changed, till he is working in the organization.

⇨ The primary key of a relation can be said to be a minimal super key

2.1.3 Candidate Key

A candidate Key is an attribute or set of attributes that uniquely identifies a record. These attributes or combinations of attributes are called candidate keys. In such a case, one of the candidate key is chosen to be a primary key. The remaining candidate keys are called alternate keys.

⇨ *There is only one primary key in a table. But there can be multiple candidate keys.*

2.1.4 Composite Key

In many cases, as we design a database, we will have tables that will use more than one column as part of the primary key. These are called Composite Keys or (concatenated keys). In other words, when a record cannot be uniquely identified by a single field, in such cases a composite key is used. A composite key is a group of fields that are combined together to uniquely identify a record.

2.1.5 Secondary Key

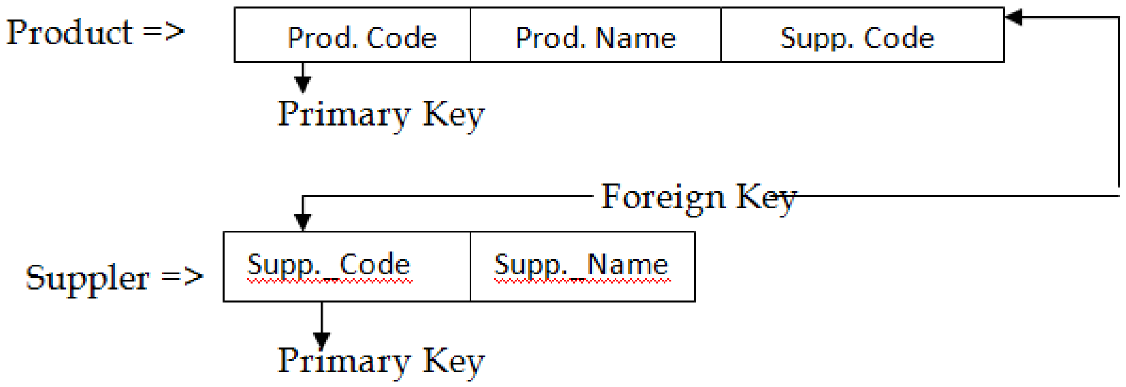
A Secondary Key is an attribute or combination of attributes that may not be a candidate key but classifies the entity set on a particular characteristic. For example, the entity set EMPLOYEE having the attribute Department, which identifies by its value which means all instances of EMPLOYEE who belong to a given department.

More than one employee may belong to a department, so the Department attribute is not a candidate key for the entity set EMPLOYEE, since it cannot uniquely identify an individual employee. However, the Department attribute does identify all employees belonging to a given department. Hence, It can be considered a a secondary key.

2.1.6 Foreign Key

In a relation, the column whose data values correspond to the values of a key column in another relation is called a Foreign Key. The Supp_code key is a Foreign Key as seen in Figure 2.1.

⇨ In a relational database, the foreign key of a relation, may be the primary key of another relation.



Q.2.2 What are Relationships?

Ans 2.2 RELATIONSHIPS

Entities may have several relationships among themselves. Whenever an attribute of one entity refers to another entity, three exists a relationship between the two entities.

Q2.3 What is Normalilzation?

Ans Normalization is the name given to the process of simplifying the relationship among data elements in a record. Normalization replaces a collection of data in record structure by another record design which is simple, more predictable and therefore more manageable.

The goal of relational database design is to generate a set of relation schemes that allows us to store information without any redundant (repeated) data. It also allows us to retrieve information easily and more efficiently.

The first step towards Normalization is to convert E-R model into Tables or Relations. The next steps are to examine the tables for redundancy and if necessary, change them to non-redundant forms. This non-redundant model is when converted to a database definition, which achieves the objective of the database Design Phase.

2.3.1 Need for Normalization

Normalization reduces redundancy. Redundancy is the unnecessary repetition of a field. It can cause problems with storage, retrieval and updation of data. Redundancy can lead to:

(a) Inconsistencies-errors are more likely to occur when facts are repeated.

(b) Update anomalies-inserting, modifying and deleting data may cause inconsistencies. Inconsistency occurs when we perform updation or deletion of data in one relation, while forgetting to make corresponding changes in other relations.

⇨ *During the process of normalization, we can identify dependencies, which do cause problems when deleting or updating. Normalization helps in simplify the structure of the tables.*

A fully normalized record consists of:

(a) A primary key that identifies that entity.

(b) A set of attributes that describe that entity.

In the process of normalization, data are grouped in the simplest possible way so that changes can be made later with minimum impact on the data structure. Various steps in normalization process are described later in this chapter.

Q.2.4 Define First Normal Form.

Ans First Normal Form (1NF)

A table is in the First Normal Form when it contains no repeating groups. The repeating columns or fields present in a UN normalized table are removed from the table and put into separate table or tables. These tables are dependent on the parent table from which it is derived. The key to this table must also be a part of the parent table, so that the parent table and the derived tables can be related to each other.

Isolate repeating groups from an entity because they are easier to process separately, from rest of the entity. Figure 2.2 shows an un normalized table structure.

⇨ *When a table has no repeating groups, it is said to be in first normal form (1 NF). That is, for each dell in a table (one row and one column), there can be only one value. This value should be atomic in the sense that it cannot be decomposed into smaller pieces.*

As seen in Figure 2.2, the first four attributes (Employee number, Employee name, Store branch and Department) are virtually constant. The remaining three attributes (Item number, Item description, and Sale price) contain data that change and are repeated with different sales persons. Therefore, the repeating group should be separated from the entity "salesperson".

The normalized file is shown in Figure 2.3. It consists of two files:

Sales person				**Sales**		
Employee Number	**Employee Name**	**Store Branch**	**Department**	**Item Number**	**Item Description**	**Sale Price (Rs.)**
21130680	**Anand K**	**Downtown**	**Hardware**	**TR10**	**Router**	**35.00**
				SA 1	**Saw**	**19.00**
				PT 6	**Drill**	**21.00**
				AB16	**Lawanmover**	**245.00**
30142101	**Zadoo S**	**Dadeland**	**Home Appliance**	**TT 1**	**Humidfier**	**114..00**
				DS10	**Dishwasher**	**262.00**
41984620	**Balwant**	**Cutter Point**	**Auto parts**	**MC16**	**Snow tire**	**85.00**
				AC146	**Alternator**	**65.00**
				BB100	**Battery**	**49.50**
61204721	**Bhagwan**	**Fashion spot**	**Men's clothing**	**HS10**	**Suit**	**215.00**

Key

Employee Number	Employee Name	Store Branch	Department
21130680	Anand K	Downtown	Hardware
30142101	Zadoo S	Dadeland	Home appliances
41984620	Balwant	Cutter point	Auto parts
61204721	Bhagwan	Fashion poiont	Men's clothing

Salesperson Data File

Employee Number	Item Number	Item Description	Sale Price (Rs.)
21130680	TR 10	Router	35.00
21130680	SA 1	Saw	19.00
21130680	PT 6	Drill	21.00
21130680	AB 16	Lawnmover	245.00
30142101	TT 1	Humudfier	114.00
30142101	DS 10	Dishwasher	262.00
41984620	MC 16	Snow tire	85.00
41984620	AC 146	Alternator	65.00
41984620	BB 100	Battery	49.50
61204721	HS 10	Suit	215.00

Salesperson Item File

(a) The salesperson data file with employee number as the primary key.

(b) The salesperson item file with employee number and item number as new attributes. These two attributes are added to relate the records in this file to the salesperson data file. The two attributes are used together for accessing data. Therefore, two keys are used together and such a key is called a concatenated key.

Q2.5 Define the Functional Dependencies.

Ans Functional dependencies play an important role in differentiating good database design from not so good database design. Functional dependencies are the consequence of the interrelationships among attributes of an entity represented by a relation. Alternatively, it may be due to the relationship between entities that are also represented by a relation. Given a relation R, attribute Y of R is functionally dependent on attribute X of R if and only if each X-value in R is associated with it precisely one Y-value in R.

⇨ *A functional dependency is denoted by X → Y, between two sets of attributes X and Y.*

To understand functional dependencies in a better way. Let us see the database containing information concerning suppliers (S) and parts (P). The suppliers and parts are uniquely identified by Supplier number (SNo) and Part no (PNo).

	S.No.	Name	Status	City
S=>	S1	Shyam	20	Bombay
	S2	Ram	10	Calcutta
	S3	Amit	30	Calcutta
	S4	Chirag	20	Delhi
	S5	Ramesh	30	Calcutta

	PNo.	Name	Colour	Weight	City
P=>	P1	Nut	Red	12	Bombay
	P2	Bolt	Green	17	Calcutta
	P3	Screw	Blue	17	Goa
	P4	Screw	Red	14	Bombay
	P5	Handle	Blue	12	Bombay
	P6	Wire	Red	19	Delhi

In the supplier and parts databases, attributes, Name, Status and City of relation S are each functionally and City of relation S are each functionally dependent on attribute SNo, because given a particular value for SNo, there exists precisely one corresponding value for each of Name, Status and City, Syimbolically we can write:

S.SNo - S. Name
S.SNo.- S. Status
S.SNo.- S. City

The statement "S. SNo.-S.City" is read as "attribute S. City is functionally dependent on attribute S. S. No.".

The statement "S. SNo. – S. (Name, Status, City)" can be similarly interpreted if we consider the combination (Name, Status, City) as composite key of relations. Functional dependency can be shown diagrammatically as in Figure 2.4

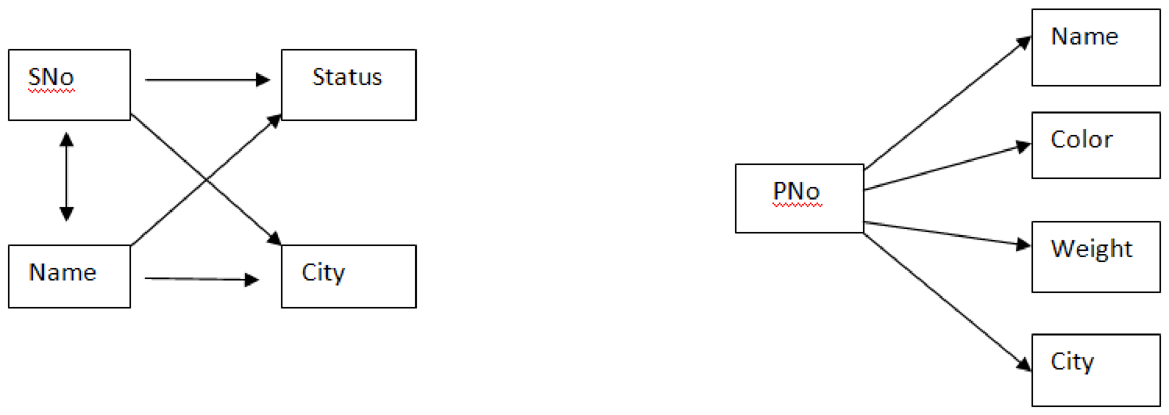


Figure 2.4 Functional dependencies in relation S an P

⇨ *Note that in relation S, we have both S.SNo-Name and Name-S.SNo because Name is an alternate key for relation S.*

Similarly, in the Student (Name, Address, Subject, Grade) relation, following functional dependencies should hold.

Name - Address
Name Subject - Grade

The underlying semantics of these two functional dependencies are that the address of a Student is unique and in each subject a student gets a unique grade. Consider an Employee database as Employee (Emp_Code, Emp_Nam,e, Dept, Grade, Salary, Age, Address) relation.
The following functional dependencies hold:

F1 : Emp_Code – Emp_Name;
(each employee has a unique Emp_Code)

F2 :Emp_Code – Dept;
(an employee can work in one department only)

F3 : Emp_Code Grade Age – Salary;
(employee's salary depends on his age and grade)

F4 :Emp_Code – Age;
(each employee has a unique age)

F5 : Emp_Code – Address;
(each employee has unique address)
In this relation Emp_Code is not functionally dependent on Salary or Age, because more than one employee can have the same salary and can be of the same age.

2.5.1 Full Functional Dependency

X→ Y is a fully functional dependency because removal any attribute A from X would into the cancellation of dependency. In other words, for any attribute A∈ X, it does not functionally determine Y. For example, in the relation S, the attribute CITY is functionally dependent on the composite attribute (SNo, Status). However CITY is not fully functionally dependent on this composite attribute because it is also functionally dependent on SNo. (If Y is functionally dependent on X but not fully so, then X must be composite).

⇨ *When all non-key attributes are dependent on the key attribute, it is called full functional dependency (See Figure [2.5])*

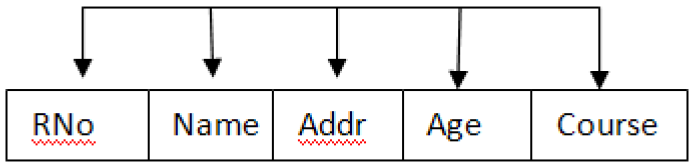


Figure 2.5 Concept of full functional dependency

In the above example non key attributes (Name, Address, Age and Course) are dependent on key attribute RNo. Here RNo. Stands for Roll Number.

2.5.2 Partial Dependency

A functional dependency X→ Y is a partial dependency if some attributes A∈ X can be removed from X and the dependency still holds. Partial dependency in a record type occurs when some non-key attribute depends on the key attribute, and the remaining non-key attributes depend on key attribute and on one or more non-key attributes. In other words, all the non-key attributes are not dependent on the key attribute. There is a partial dependency of non-key attributes either on the key attribute or on the non-key attribute. (See Figure 2.6)

In Figure 2.6, non-key attributes Name, Address and Age are dependent on RNo, and Date of Comp (Date of Completion) i.e. non-key attribute depends on RNo (key attribute) as well as on Course (non-key attribute).

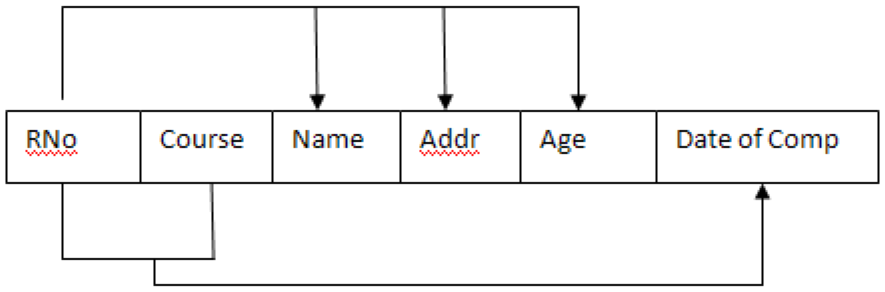


Figure 2.6 Concept of partial dependency

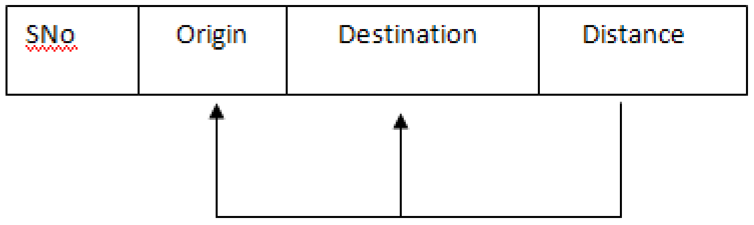


Figure 2.7(a) Concept of transitive dependency

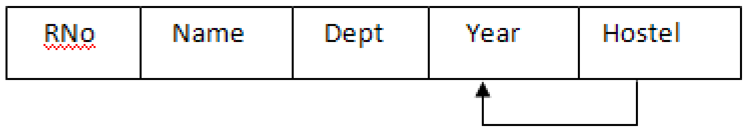


Figure 2.7(a) Concept of transitive dependency

2.5.4 Multivalued Dependency

Functional dependencies rule out certain tuples from being in a relation. If A→ B, then we cannot have two tuples with the same A value but different B values. Multivalued dependencies, on the other hand, do not rule out the existence of certain tuples, which have multiple dependencies. Instead, they require that other tuples of a certain form be present in the relation.

Multivalued dependencies are a consequence of first normal form. First normal form does not allow an attribute in a tuple to have more than one value. If we have two or more multivalued independent attributes in the same relation scheme, then we would get into a problem of having to repeat every value of one of the attributes with every value of the other attribute to keep the relation instances consistent. This constraint is specified by a multivalued dependency.

⇨ *Functional dependencies are also referred to as equality generating dependencies, and multivalued dependencies are also referred to as tuple generating dependencies.*

For Example, consider the following relation EMP:

ENAME	PNAME	DNAME
Rajan	X	Ganesh
Ramesh	Y	Dinesh
Ram	X	Babu
Arjun	Y	Bhaskar

A tuple in this EMP relation represents the fact that an employer whose name is ENAME works on the project whose name is PNAME and has a dependent whose name is DNAME. An employee may work of several projects and may have dependents. Also not that the employee, projects and dependent are not directly related to one another.

To keep the tuples in the relation consistent, we must keep a tuple to represent every combination of an employee's dependent and an employee's project. This constraint is specified as a multivalued dependency on the EMP relation. Informally, whenever two independent 1:N relationships A:B and A:C are mixed in the same relation, then multivalued dependencies may arise. Consider the Course (Course#, Teacher, Student, Time, Room) relation as given in Figure 2.8. Consider an instance of multivalued dependencies.

The meanings of these two multivalued dependencies are that the same teacher is teaching a particular course irrespective of time or room in which the course is held. Moreover, students registered for a course are not determined by the time or room where the course is held.

m1 : Course# →→ Time, Room
m2 : Course# →→ Student

Course#	Teacher	Student	Time	Room
BS03	Jain, S	Beena	M3	3
BS03	Jain, S	Aman	M3	3
PC02	Singh,S	Bipin	M5	2
PC02	Singh,S	Aman	M5	2
PC02	Singh,S	Kamal	M5	2
BS03	Bose, V	Vivek	Th7	1
BS03	Bose, V	Beena	Th7	1
BS03	Bose, V	Kamal	Th7	1
PC02	Arora, K	Bipin	F7	1
POC02	Arora, K	Aman	F7	1
PC02	Arora, K	Kamal	F7	1

Figure 2.8 An instance of the multivalued dependencies in relation (Course #, Teacher, Student, Time, Room)

The meanings of these two multivalued dependencies are that the same teacher is teaching a particular course irrespective of time or room in which the course is held. Moreover, students registered for a course are not determined by the time or room where the course is held.

2.5.5 Join Dependency

Join dependency is a constraint, similar to a Functional Dependency or a Multivalued Dependency. It is satisfied if and only if the relation concerned is the join of certain number of projections. And therefore, such a constraint is called a join dependency.

We now consider a special class of join dependencies which help to capture data dependencies present in a hierarchical data structure. For example in the NURS, HOME database shown in Figure 2.8 data has a hierarchical organization that information regarding wards and patients currently admitted to a ward depend only

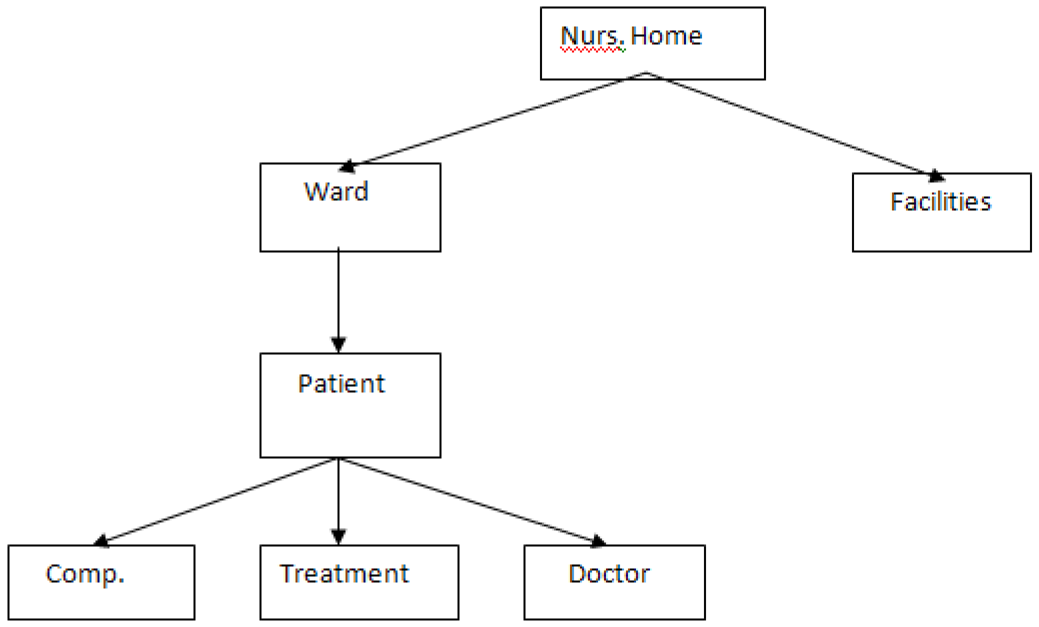


Figure 2.9 Hierarchical representation of a NURS_HOME database

On the Nurs_home but not the facilities present in that hospital (and vice versa). Since a Nurs_home can have multiple wards, functional dependencies are not adequate to describe the data dependency among NURS_HOME and WARDS or FACILITIES. In this case, Multivalued dependencies, NURS_HOME→→ WARD or NURS_HOME→→ FACILITIES hold.

Using first order hierarchical decomposition (POHD) would enable us to represent data dependencies present in a hierarchical data structure in a more natural way.
The NURS_HOME database shown in Figure 2.9 when decomposed, has the following representations.

Fh_1	:	NURS_HOME	:	WARD|FACILITIES
Fh_2	:	NURS_HOME	:	PATIENT
Fh_3	:	NURS_HOME		

WARD, PATIENT :
COMPLAINTS | TREATMENT | DOCTOR

Thus we can store NURS_HOME database as the lossless join of
NURS_FACILITY (NURS_HOME, FACILLITY)
NURS_WARD, (NURS_HOME, WARD, PATIENT,
COMPLAINTS, TREATMENT, DOCTOR)
Relations. We can use fh_2 and fh_3 to further decompose NURS_WARD relation.

2.6 SECOND NORMAL FORM (2NF)

A table is in the Second Normal Form if all its non-key fields are fully dependent on the whole key. Those that do not depend upon the combination key, are moved to another table on whose key they depend on. Structures which do not contain combination keys are automatically in the second normal form.

The second normalization makes sure that each non-key attribute depends on a key attribute or on a composite key. Non-key attributes that do not meet this condition are split into simpler entities. In Figure 2.10, each attribute in the salesperson data file depends on the primary key "Employee#". In the salesperson item file, the attribute "Sales price" depends on a composite key ("Employee#" and "Items#"). Note that the sale price is firmly related to the salesperson number and the item number of the sale. Also, the attribute "Item description" tags to "Item#", which is part of the composite key. "Item#" is not related in any way to the "Employee" filed. This causes several concerns. An employee transfer would make it difficult to maintain records because the sales information would be dropped when the salesperson leaves the department. This is because sales information (Item#, Sale price) is linked with "Employee#" in the salesperson item file.

To solve such a problem, we create new independent tables for "Item description" and "Sales price". In one file, we create the item description attribute with item number keys from the Salesperson Item file. The remaining attributes (Employee number, Item number, and Sales price) become the second table or file. (See Figure 2.10).

The creation of the second table offers several benefits. These are:

(a) Sales items can be added without being tagged to a specific salesperson.

(b) If the item changes we need to change only the item file.

(c) If a salesperson leaves the department, it would have no direct effect on the status of the items sold.

⇨ *A table is in Second normal form (2NF) if every non-key column depends on the entire key (not just part of it). This issue arises only for composite keys (with multiple columns).*

2.7 THIRD NORMAL FORM (3NF)

A table is said to be in the Third Normal Form, if all the non-key fields of the table are independent of all other non-key fields of the table,

In Figure 2.10 we can observe that there is further room for improvement. In the salesperson data file, the attribute "Store branch" is tagged to the primary key "Employee#" while the attribute "Department".

Employee #	Employee Name	Store Branch	Department
21130680	Anand K	Downtown	Hardware
30142101	Zadoo S	Dadeland	Home appliance
41984620	Balwant	Cutter point	Auto parts
61204721	Bhagwan	Fashion spot	Men's clothing

1

Salesperson Data File

2

Employee #	Item #	Sale Price (Rs.)
21130680	TR 10	35.00
21130680	SA 1	19.00
21130680	PT 6	21.00
21130680	AB 16	245.00
30142101	TT 1	114.00
30142101	DS 10	262.00
41984620	MC 16	85.00
41984620	AC146	65.00
41984620	BB100	49.50
61204721	HS 10	215.00

Salesperson Item File

Item #	Item Description
TR10	Router
SA 1	Saw
PT 6	Drill
AB16	Lawnmover
TT 1	Humidfier
DS10	Dishwasher
MC16	Snow tire
AC146	Alternator
BB100	Battery
HS10	Suit

Item File

Figure 5.10 Second normalization

Which is a non-key? Attribute is related to “Store branch”, which is another non_key attribute. Making “Store branch a key attribute requires isolating “Department” along with “Store branch and placing them a new table as shown in Figure 2.11

Note that, after completing the third normalization, we can store branch information independent of the salespersons in the branch. We can also make changes in the “Department” without having to update the record of the employee in it. In this respect normalization simplifies relationships and provides logical links between files without losing information.

One inherent problem with normalization is data redundancy. For store branch, the system goes through three steps:

(a) Computes total sales for each salesperson from the salesperson item file.

(b) Goes to the employee data file to look up the store branch to which the sales person is assigned.

(c) Accumulates each salesperson’s sales in a specified field in the store branch file.

This procedure is repeated for each salesperson in the file. Figure 2.12 illustrates the processing cycle for salesperson 21130680.

⇨ *A relation is in the third normal form if it is second normal form and no non-prime attribute is functionally dependent on other non-prime attributes.*

Q 2.8 Define BCNF (BOYCE-CODD NORMAL FORM).

Ans Boyce-Codd Normal Form (BCNF) was proposed as

1

Employee #	Employee Name	Store Branch
21130680	Anand K	Downtown
30142101	Zadoo S	Dadeland
41984620	Balwant	Cutter point
61204721	Bhagwan	Fashion Spot

Store Branch	Department
Downtown	Hardware
Dadeland	Home appliances
Cutter point	Auto parts
Fashion Spot	Men's clothing

1

Salesperson Data File
Store Branch File

2

Employee #	Item #	Sale Price (Rs.)
21130680	TR 10	35.00
21130680	SA 1	19.00
21130680	PT 6	21.00
21130680	AB 16	245.00
30142101	TT 1	114.00
30142101	DS 10	262.00
41984620	MC 16	85.00
41984620	AC146	65.00
41984620	BB100	49.50
61204721	HS 10	215.00

Salesperson Item File

Item #	Item Description
TR 10	Router
SA 1	Saw
PT 6	Drill
AB 16	Lawnmover
TT 1	Humidifier
DS 10	Dishwasher
MC 16	Snow tire
AC146	Allternator
BB 100	Battery
HS 10	Suit

Item File

Figure 2.11 Third normalization

a simpler form of Third Normal Form (3NF), but it is much more strict than 3NF, meaning that every relation in BCNF is also in 3NF. However, a relation in 3NF is not necessarily be in BCNF.

Consider a relation scheme in third normal form that has a number of overlapping composite candidate keys. In particular consider the relation GRADE (Name, Student#, Course, Grade) of Table 2.1. Here the functional dependencies are (Name Course→ Grade, Student# Course→ Grade, Name→ Student# , Student# → Name). Thus, each student has a unique name and a unique student number. The relation has two candidate keys, (Name, Course) and (Student#, Course). Each of these keys is a composite key and contains a common attribute Course. The relation scheme satisfies the criterion of the third normal from relation, i.e., for all functional dependencies $X \rightarrow A$ in GRADE.

The problem in the relation GRADE is that it had two overlapping candidate keys. In the Boyce Codd normal from (BCNF), which is stronger than the third normal form, the intention is to avoid the above anomalies. This is done by ensuring that for all non-trivial functional dependencies implied by the relation, determine the functional dependencies that involve a candidate keys.

The relation GRADE of Table 2.1 is not in BCNF because the dependencies Student# $\rightarrow$ *Name and Name* $\rightarrow$ Student# are non-trivial and their determinants are not super keys of GRADE.

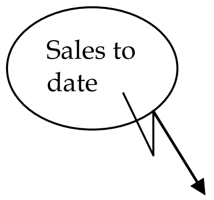
Q 2.9 Define FOURTH NORMAL FORM (4NF).

Ans In order to avoid undesirable updating problem in presence of multi-valued dependencies, Fourth Normal Form was introduced as an extension of Boyce Codd normal form.

Fourth Normal Form eliminates cases in which the composite key of a record type contains two or more data items that are independent, multi-valued facts of an entity.

The fields are independent if there are no combinations that are logically related. In the example given in Figure 2.13, skill and languages are multi-valued (many different skills and many different languages), but there is no logical dependency between a language and a skill.

To put in Fourth Normal Form, the data must be separated by creating two separate tables.



Employee #	Employee Name	Store Branch
21130680	Anand K	Downtown
30142101	Zadoo S	Dadeland
41984620	Balwant	Cutter point
61204721	Bhagwan	Fashion Spot

2 Salesperson Data File

Store Branch	Department	Total Sale (Rs.)
Downtown	Hardware	320.00
Dadeland	Home appl.	376.00
Cutter point	Auto parts	199.50
Fashion point	Men's clothing	215.00

Store Branch File

1

Employhee#	Item #	Sale Price (Rs.)
21130680	TR 10	35.00
21130680	SA 1	19.00
21130680	PT 6	21.00
21130680	AB 16	245.00
30142101	TT 1	114.00
30142101	DS 10	262.00
41984620	MC 16	85.00
41984620	AC146	65.00
41984620	BB100	49.50
61204721	HS 10	215.00

Item #	Item Description
TR 10	Router
SA 1	Saw
PT 6	Drill
AB 16	Lawnmover
TT 1	Humidfier
DS 10	Dishwasher
MC 16	Snow tire
AC146	Alternaor
BB100	Battery
HS 10	Suit

Table 2.1 The GRADE relation

Name	Student #	Course	Grade
James	23714539	353	A
Neelam	42717390	329	A
James	23714539	329	in prog
Mohan	38815183	456	C
Dilip	37116259	293	B
Deepak	82317293	491	C
Deepak	82317293	353	In prog
James	23714539	491	C
Raj	11011978	353	A+
Vikas	83910827	379	in prog

⇨ *A generalization of the Boyce Codd normal form to a relation which includes the multi-valued dependencies is called Fourth Normal Form.*

Q 2.10 Define FIFTH NORMAL FORM (5NF)
Ans

The Fourth Normal Form is by no means the "ultimate" normal form. As we saw earlier, multi-valued dependencies help us understand and tackle some forms of repetition of information that cannot be understood in terms of functional dependencies. There are types of constraints called join dependencies that generalize multi-valued dependencies, and lead to another normal form called project-join normal form (PJNF). The PJNF is called Fifth Normal Form. There is a class of even more general constraints, which leads to normal form called domain-key normal form.

A practical problem with the use of these generalized constraints is that they are not only hard to reason with, but there is also no set of sound and complete inference rules for reasoning about the constraints. Hence, PJNF and domain key normal form are used very rarely.

Employee	Skill	Language

must be separated as

Employee	Skill

Employee	Language

Figure 2.13 Fourth Normal Form

⇨ *The Concept of Project-join Normal form (PJNF) is an extension of the Fourth Normal form definition when join dependencies are also considered.*

2.11 DOMAIN-KEY NORMAL FORM

The process of normalization and the process of discovering undesirable dependencies were carried through Fifth Normal Form (5NF) as a meaningful design activity. However, it has been possible to define stricter normal forms that take into account additional types of dependencies and constraints. The idea behind domain-key Normal Form (DKNF) is to specify the ultimate normal form that takes into account all possible types of dependencies and constraints.

The domain key normal form (DKNF) does not exhibit insertion and deletion anomalies. However, unlike the other normal forms, DKNF is not defined in terms of functional dependency, multi-valued dependency, or join dependency. The central requirements in DKNF are the basic concepts of domain, keys and general constraints. A domain constraint is a statement that the values of a certain attribute lie within some prescribed set of values. Key constraints are a statement that certain attribute or attribute combination is a candidate key. A general constraint is expressed as a simple statement or predicate and specifies some special requirement. Each tuple of a relation must specify this predicate i.e. general constraint for it to be a valid tuple.

- ⇨ *A relation is in DKNF, if every general constraint can be inferred from the knowledge of the attributes in the scheme, their underlying domains, and the sets of attributes that form the keys.*

An insertion anomaly in the case of DKNF occurs when a tuple is inserted in a relation and the resulting relation violates one or more general constraints. Similarly, a deletion anomaly occurs when a tuple from a relation is deleted and the remaining relation violates one or more general constraints. We show these dependencies and general constraints in the following example.

Example 1

Consider the relation scheme TRANSCRIPT (SNo, Course, Grade), Suppose the attributes SNo and Course are numeric, 8 and 3 digits long, respectively. The attribute Grade is a letter grade and could be A, B, C D, P, F. The general constraint is that for Course900 through 900, the Grade can only be A, B, C, D, F. The domain constraints for this relation are the following:

SNo is required to be 8 digits long, Course is 3 digit long, and Grade has to be from the set (A, B, C, D, F).

The key constraint for the relation is that no two tuples can exist with the some values for the key attributes, which are SNo and Course. Obviously,

SNo. Course$\rightarrow$ Grade.

Finally, the general constraint can be expressed by the following:

If Grade > 900 then Grade $\in$ [P, F] ELSE Grade $\in$ [A, B, C, D, F]

- ⇨ *The DKNF is considered to be the highest form of normalization, since all insertion and deletion anomalies are eliminated and all general constraints can be verified by using only the domain constraints and key constraints.*

The problem with this relation is that a tuple such as (12345678, 991, A), which satisfies all the domain constraints and key constraints, can be inserted in the relation TRANSCRIPT of Table 2.2. However, since the tuple does not satisfy the general constraint, the relation TRANSCRIPT becomes illegal after the insertion.

Table 2.2 The TRANSCRIPT relation

S.No.	Course	Grade
23714539	353	A
42717390	329	A
23714539	928	P
38815183	456	F
37116259	293	B
82317293	491	C
82317293	953	F
23714539	491	C
11011978	353	A
83910827	379	P

Example 2

The TRANSCRIPT relation discussed above can be decomposed into the following relations:
TRANSCRIPT_REGULAR (SNo, Course, Grade) with the domain constraints (SNo being 8 digit, Course being 3 digit in the range 000 through 900, and Grade in the set [A, B, C, D, F]. The key as before is SNo Course.
TRANSCRIPT_SPECIAL (SNo, Course, Grade) with the domain constraints (SNo being 8 digit, Course being 3 digit in the range 000 through 999, and Grade in the set [P, F]. The key as before is SNo Course

2.12 DENORMALIZATION

In some exceptional cases, database designers use the redundancy to improve performance for specific applications. They select such a scheme that has redundant information that means it is not normalized.

For example, suppose that the name of an account holder has to be displayed along with the account number and cash balance, every time the account is accessed. In our normalized scheme, this requires a join of account with depositor. One alternative is to create a relation containing all the attributes of account and depositor. This makes displaying the account information faster. However, the balance information for an account is repeated for every person who owns the account, and all copies must be updated by the application, whenever the account balance is updated. This process of taking a normalized scheme and making it non-normalized is called renormalization. Designers of database use it to tune Performance of systems that require time critical operations.

A better alternative is to use views. A materialized view is a view whose result is stored in the database, and reflected in the data when the relations used in the view are updated. Like denormalization, using materialized view does have space and time overheads, but it has the advantage that keeping the view up-to-date is the job of database system, not the application programmer.

2.13. CASE STUDY 1

Normalize the following data of CUSTOMER table to 2NF using appropriate relation. Give brief explanation for each level.

XYZ LTD.

Cust_no	001
Cust_name	Dev and Co. Ltd.
Addr	DWARKA, SECTOR- , Pl. No. 2
Order_no	2240
Order_date	5/11/02
Deli_date	15/11/02

Item_no	Desc	Rate	Qty.
001	Floppy disk (3 ¼”)	170	5
024	Printer (Inkjet)	5500	2
035	CD	450	5
039	Keyboard	1000	3

Solution

Step 1

Firstly, we have to write down the data in the un normalized form. Without normalization, the data appears as follows:

Chapter 2

Architecture of DBMS

Q.1 Describe the three levels of data abstraction?

Ans: The are three levels of abstraction:

1. **Physical level:** The lowest level of abstraction describes how data are stored.
2. **Logical level:** The next higher level of abstraction, describes what data are stored in database and what relationship among those data.
3. **View level:** The highest level of abstraction describes only part of entire database.

Q.2 Define the Overall System Structure of Database Management System.

Ans.: Overall System Structure :

a) Database systems are partitioned into modules for different functions. Some functions (e.g. file systems) may be provided by the operating system.

b) **Components include :**

- **File Manager** manages allocation of disk space and data structures used to represent information on disk.
- **Database Manager** : The interface between low-level data and application programs and queries.

- **Query Processor** translates statements in a query language into low-level instructions the database manager understands. (May also attempt to find an equivalent but more efficient form.)
- **DML Precompiler** converts DML statements embedded in an application program to normal procedure calls in a host language. The precompiler interacts with the query processor.
- **DDL Compiler** converts DDL statements to a set of tables containing metadata stored in a data dictionary.

In addition, several data structures are required for physical system implementation :

- **Data Files :** store the database itself.
- **Data Dictionary :** stores information about the structure of the database. It is used **heavily**. Great emphasis should be placed on developing a good design and efficient implementation of the dictionary.
- **Indices :** provide fast access to data items holding particular values.

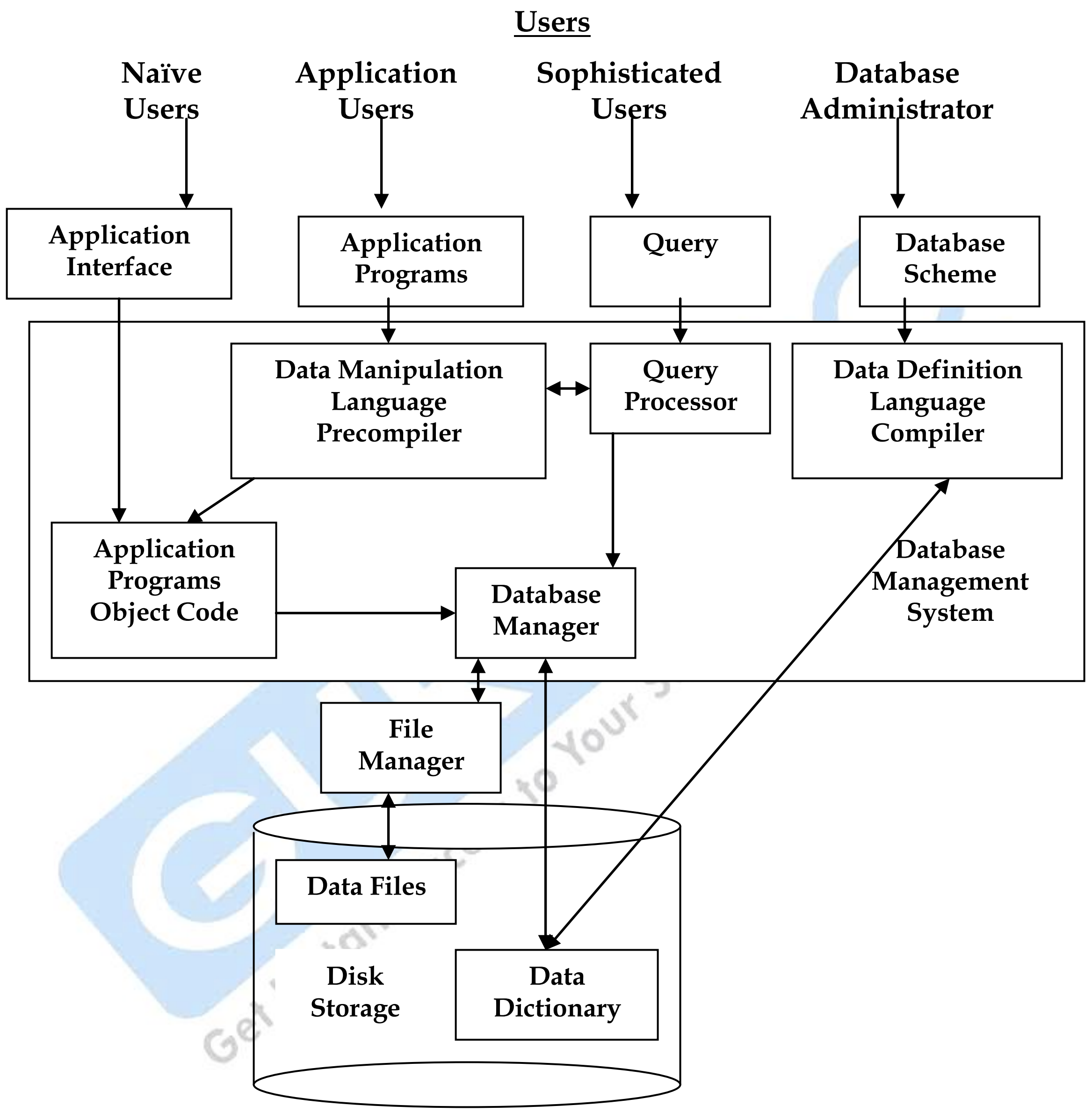


Figure : Database System Structure

Chapter 3

Introduction to Data

Q.1 Explain Hierarchical, Network and Relational, Database Models

Ans: Database models continue to evolve as the information management needs of organizations become more complex. From flat files to **relational databases**, the growing demands on data integrity, reliability and performance of database management systems (DBMS), has shaped the design of databases and their underlying models. In this document, three database models are discussed comparing and contrasting their major features. The three models in order of discussion are Hierarchical, Network and Relational database models.

Hierarchical Database Model

The hierarchical model is the oldest of the three models discussed here. This model is an improvement of the flat-file database system since it employs a simple data relationship scheme. The relationships in the hierarchical model are child/parent relationships. The name "hierarchical" is derived from one major restriction on the child/parent relationships, that is, although a parent entity can have several child entities, a child entity can only have one and only one parent. For this reason all the relationships form a hierarchy that traces back to one root. In fact, this model is often visualized as an upside down tree, where the entity at the top is seen as a root and as such all other entities sprout from the root.

As shown in the diagram, one major problem with the hierarchical model is the increased risk of data inconsistency. In the case above, a separate "Customers" table must exist for every product line due to the fact that a child entity cannot have more than one parent. However, there is a great chance that there are many customers who purchased more than one type of product.

Consequently, information about those customers must exist in more than one table causing data redundancy. Another problem with the hierarchical model is the inflexibility of the model. For example, in the diagram above, the database is restricted to three product lines. Adding a new product line would require redesigning the database since all the relationships must be predefined. Finally, another problem with the hierarchical model is in the child/parent relationship restriction. Every child must have a parent. Therefore, in the example above, it is impossible to add new customers who have not bought any new products yet. To overcome some of the limitations of the hierarchical model, the network model was created.

Network Database Model

The network model is an improvement to its predecessor, the hierarchical model. In the network model, a child entity can have more than one parent. Therefore, the design can be visualized as several inverted trees interconnected by branches as opposed to the single inverted tree characteristic of the hierarchical model

Q.2 Difference between hierarchical data model or network data mode and relational data model.

Ans: **1.The hierarchical data model :**The hierarchical data model organizes data in a tree structure. There is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. Data in a series of records, which have a set of field values attached to it. It collects all the instances of a specific record together as a record type. These record types are the equivalent of tables in the relational model, and with the individual records being the equivalent of rows. To create links between these record types, the hierarchical model uses Parent Child Relationships. These are a 1:N mapping between record types. This is done by using trees, like set theory used in the relational model, "borrowed" from maths.

2: The network data model :

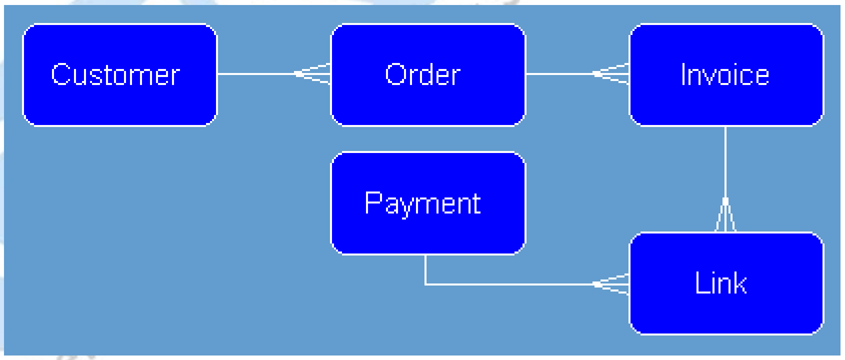
The popularity of the network data model coincided with the popularity of the hierarchical data model. Some data were more naturally modeled with more than one parent per child. So, the network model permitted the modeling of many-to-many relationships in data.

3: The relational model:
A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of relational database:
1. Values Are Atomic
2. Each Row is Unique
3. Column Values Are of the Same Kind
4. The Sequence of Columns is Insignificant 5. The Sequence of Rows is Insignificant
5. Each Column Has a Unique Name

Q.3 Explain ER Diagrams and Notations.
Ans:

- Introduction



Entity relationship diagramming is a technique that is widely used in the world of business and information technology to show how information is, or should be, stored and used within a business system.

The success of any organization relies on the efficient flow and processing of information.

In this example information flows around the various departments within the organization. This information can take many forms, for example it could be written, oral or electronic.

Here is an **example** of the sort of information flows that you might be analyzing:

The general manager regularly communicates with staff in the sales and marketing and accounts departments by using e-mail. Orders received by sales and marketing are forwarded to the production and accounts departments, for fulfillment and invoicing. The accounts department forward regular written reports to the general manager, they also raise invoices and send these to the customers.

Data modeling is a technique aimed at optimizing the way that information is stored and used within an organization. It begins with the identification of the main data groups, for example the invoice, and continues by defining the detailed content of each of these groups. This results in structured definitions for all of the information that is stored and used within a given system.

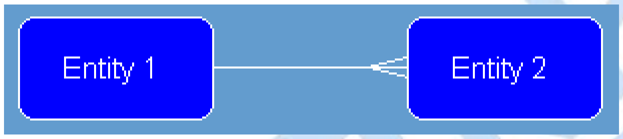
The technique provides a solid foundation for systems design and a universal standard for system documentation. Data modeling is an essential precursor to analysis & design, maintenance & documentation and improving the performance of an existing system.

Entity Relationship Diagram - Diagram Notation

Entity relationship diagramming uses a standard set of symbols to represent each of these defined data groups and then proceeds by establishing the relationships between them. The first of these symbols is the soft-box entity symbol.

An entity is something about which data will be stored within the system under consideration. In this example the data group invoice can be identified as a system entity.

The other main component on a data model is the relationship line. A Relationship is an association between two entities to which all of the occurrences of those entities must conform.



The relationship is represented by a line that joins the two entities, to which it refers. This line represents two reciprocal relationships:That of the first entity with respect to the second, and that of the second entity with respect to the first.

Entity relationship diagramming is all about identifying entities and their relationships and then drawing a diagram that accurately depicts the system. This applies equally to the design of a new system or the analysis of an existing one.

The end result of entity relationship diagramming should be a clear picture of how information is stored and related within a proposed, or existing, system.

Entity Relationship Diagram - Entities

Here, we illustrate the concept of an entity, which can be applied to almost anything that is significant to the system being studied. Some examples of information systems and their entities are listed below:

Banking system: Customer, Account, Loan.
Airline system: Aircraft, Passenger, Flight, Airport.

An entity is represented by a box containing the name of that entity.
A precise definition of 'entity' is not really possible, as they even vary in nature.
For example, in the airline system, whilst an aircraft is a physical object (entities often are) a flight is an event and an airport is a location. However entities are nearly always those things about which data will be stored within the system under investigation.

Note that entities are always named in the singular; for example: customer, account and loan, and not customers, accounts and loans.

This course uses symbols that are standard in the IT industry. This uses the soft- box symbol shown to represent an entity. If a site uses a different symbol set, this is not a problem, as entity relationship diagramming techniques are the same regardless of the symbols being used.

Entity Relationship Diagram - Entity Types & Occurrence
Similar entity occurrences are grouped together and collectively termed an entity type. It is entity types that are identified and drawn on the data model. An entity occurrence identifies a specific resource, event, location, notion or (more typically) physical object.

In this course the term 'entity' is, by default, referring to entity type. The term entity occurrence will be specifically used where that is relevant. Each entity has a data group associated with it. The elements of the data group are referred to as the 'attributes ' of the entity. The distinction between what is an attribute of an entity and what is an entity in its own right is often unclear. This is illustrated shortly.

Entity Relationship Diagram - Entity Naming
Entity names are normally single words and the name chosen should be one familiar to the users. The entity name can include a qualifier in order to clarify their meaning. However, if different names are currently used to describe a given entity in different areas of the organization then a new one should be chosen that is original, unique and meaningful to all of the users.

For example, the terms 'signed contract', 'sale' and 'agreement' might be recreated as the entity 'completed'.

Conversely an organization may be using a 'catch all' term to describe what the analyst identifies as being a number of separate entities. For example the term 'invoice' may be being used to describe 3 invoice types - each of which is, in fact, processed in a different manner.

In this case prefixing the entity names with qualifiers, is likely to be the best solution. **Notification**
The process of identifying entities is one of the most important steps in developing a data model.

It is common practice for an experienced analyst to adopt an intuitive approach to entity identification, in order to produce a shortlist of potential entities. The viability of each of these potential entities can then be considered using a set of entity identification guidelines. This should result in some of the potential entities being confirmed as entities, whilst others will be rejected.

Q.4 Explain the following terms briefly:
Attribute, Domain, Entity, Relationship, Entity set, Relationship set, one-to-many relationship, many-to-many relationship, participation constraint, Overlap constraint, Covering constraint, Weak Entity Set, Aggregation, and Role Indicator.

Ans: Term explanations:
Attribute - a property or description of an entity. A toy department employee entity could have attributes describing the employee’s name, salary, and years of service.

Domain - a set of possible values for an attribute.

Entity - an object in the real world that is distinguishable from other objects such as the green dragon toy.

Relationship - an association among two or more entities.

Entity set - a collection of similar entities such as all of the toys in the toy department.

Relationship set - a collection of similar relationships

One-to-many relationship - a key constraint that indicates that one entity can be associated with many of another entity. An example of a one-to-many relationship is when an employee can work for only one department, and a department can have many employees.

Many-to-many relationship - a key constraint that indicates that many of one entity can be associated with many of another entity. An example of a manyto-many relationship is employees and their hobbies: a person can have many different hobbies, and many people can have the same hobby.

Participation constraint - a participation constraint determines whether relationships must involve certain entities. An example is if every department entity has a manager entity. Participation constraints can either be total or partial. A total participation constraint says that every department has a manager. A partial participation constraint says that every employee does not have to be a manager.

Overlap constraint - within an ISA hierarchy, an overlap constraint determines whether or not two subclasses can contain the same entity.

Covering constraint - within an ISA hierarchy, a covering constraint determineswhere the entities in the subclasses collectively include all entities in the superclass.
For example, with an Employees entity set with subclasses HourlyEmployee and SalaryEmployee, does every Employee entity necessarily have to be within either HourlyEmployee or Salary Employee?

Weak entity set - an entity that cannot be identified uniquely without considering some primary key attributes of another identifying owner entity. An example is
including Dependent information for employees for insurance purposes.

Aggregation - a feature of the entity relationship model that allows a relationship set to participate in another relationship set. This is indicated on an ER diagram by drawing a dashed box around the aggregation.

Role indicator - If an entity set plays more than one role, role indicators describe the different purpose in the relationship. An example is a single Employee entity set with a relation Reports-To that relates supervisors and subordinates.

Q.5 What is Data Model?
Ans: A collection of conceptual tools for describing data, data relationships data semantics and constraints.

Q.6 What is E-R model?
Ans: This data model is based on real world that consists of basic objects called entities and of relationship among these objects. Entities are described in a database by a set of attributes.

Q.7 What is Object Oriented model?
Ans: This model is based on collection of objects. An object contains values stored in instance variables with in the object. An object also contains bodies of code that operate on the object. These bodies of code are called methods. Objects that contain same types of values and the same methods are grouped together into classes.

Q.8 What is an Entity?
Ans: It is a 'thing' in the real world with an independent existence.

Q.9 What is an Entity type?
Ans: It is a collection (set) of entities that have same attributes.

Q.10 What is an Entity set?
Ans: It is a collection of all entities of particular entity type in the database.

Q.11 What is an Extension of entity type?
Ans: The collections of entities of a particular entity type are grouped together into an entity set.

Q.12 What is Weak Entity set?
Ans: An entity set may not have sufficient attributes to form a primary key, and its primary key compromises of its partial key and primary key of its parent entity, then it is said to be Weak Entity set.

Q.13 What is an attribute?

Ans: It is a particular property, which describes the entity.

Chapter 4

Entity Relationship Model

Q.1 What are partial, alternate,, artificial, compound and natural key?
Ans:

1. **Partial Key:** It is a set of attributes that can uniquely identify weak entities and that are related to same owner entity. It is sometime called as Discriminator.
2. **Alternate Key:** All Candidate Keys excluding the Primary Key are known as Alternate Keys.
3. **Artificial Key:** If no obvious key, either stand alone or compound is available, then the last resort is to simply create a key, by assigning a unique number to each record or occurrence. Then this is known as developing an artificial key.
4. **Compound Key:** If no single data element uniquely identifies occurrences within a construct, then combining multiple elements to create a unique identifier for the construct is known as creating a compound key.
5. **Natural Key:** When one of the data elements stored within a construct is utilized as the primary key, then it is called the natural key.

Q.2 What's the difference between a primary key and a unique key?
Ans: Both primary key and unique enforce uniqueness of the column on which they are defined. But by default primary key creates a clustered index on the column, where are unique creates a nonclustered index by default. Another major difference is that, primary key doesn't allow NULLs, but unique key allows one NULL only.

Q.3 What are constraints? Explain different types of constraints.
Ans: Constraints enable the RDBMS enforce the integrity of the database automatically, without needing you to create triggers, rule or defaults. Types of constraints: NOT NULL, CHECK, UNIQUE, PRIMARY KEY, FOREIGN

KEY. For an explanation of these constraints see books online for the pages titled: "Constraints" and "CREATE TABLE", "ALTER TABLE"

Chapter 5

Normalization

Q.1 What is normalization?

Ans: It is a process of analysing the given relation schemas based on their Functional Dependencies (FDs) and primary key to achieve the properties (1).Minimizing redundancy, (2). Minimizing insertion, deletion and update anomalies.

Q.2 Explain Normalization.

Ans: Well a relational database is basically composed of tables that contain related data. So the Process of organizing this data into tables is actually referred to as normalization.

Q.3 What is Functional Dependency?

Ans: A Functional dependency is denoted by X Y between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuple that can form a relation state r of R. The constraint is for any two tuples t1 and t2 in r if t1[X] = t2[X] then they have t1[Y] = t2[Y]. This means the value of X component of a tuple uniquely determines the value of component Y.

Q.4 What is 1 NF (Normal Form)?

Ans: The domain of attribute must include only atomic (simple, indivisible) values.

Q.5 What is Fully Functional dependency?

Ans: It is based on concept of full functional dependency. A functional dependency X Y is full functional dependency if removal of any attribute A from X means that the dependency does not hold any more.

Q.6 What is 2NF?

Ans: A relation schema R is in 2NF if it is in 1NF and every non-prime attribute A in R is fully functionally dependent on primary key.

Q.7 What is 3NF?

Ans: A relation schema R is in 3NF if it is in 2NF and for every FD X A either of the following is true

1. X is a Super-key of R.
2. A is a prime attribute of R.

In other words, if every non prime attribute is non-transitively dependent on primary key.

Q.8 What is BCNF (Boyce-Codd Normal Form)?

Ans: A relation schema R is in BCNF if it is in 3NF and satisfies an additional constraint that for every FD X A, X must be a candidate key.

Q.9 What is 4NF?

Ans: A relation schema R is said to be in 4NF if for every Multivalued dependency X Y that holds over R, one of following is true.
1.) X is subset or equal to (or) XY = R.
2.) X is a super key.

Q.10 What is 5NF?

Ans: A Relation schema R is said to be 5NF if for every join dependency {R1, R2, ..., Rn} that holds R, one the following is true

1.) Ri = R for some i.
2.) The join dependency is implied by the set of FD, over R in which the left side is key of R.

Q.11 What is Domain-Key Normal Form?

Ans: A relation is said to be in DKNF if all constraints and dependencies that should hold on the the constraint can be enforced by simply enforcing the domain constraint and key constraint on the relation.

Q.12 What is denormalization and when would you go for it?

Ans: As the name indicates, denormalization is the reverse process of normalization. It's the controlled introduction of redundancy in to the database design. It helps improve the query performance as the number of joins could be reduced.

Chapter 6

Relation Model and DBA

Q.1 What is a Relation Schema and a Relation?

Ans: A relation Schema denoted by R(A1, A2, ..., An) is made up of the relation name R and the list of attributes Ai that it contains. A relation is defined as a set of tuples. Let r be the relation which contains set tuples (t1, t2, t3, ..., tn). Each tuple is an ordered list of n-values t=(v1,v2, ..., vn).

Q.2 What is degree of a Relation?

Ans: It is the number of attribute of its relation schema.

Q.3 What is Relationship?

Ans: It is an association among two or more entities.

Q.4 What is Relationship set?

Ans: The collection (or set) of similar relationships.

Q.5 What is Relationship type?

Ans: Relationship type defines a set of associations or a relationship set among a given set of entity types.

Q.6 What is degree of Relationship type?

Ans: It is the number of entity type participating.

Q.7 What is Relational Algebra?

Ans: It is procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation.

Q.8 What is Relational Calculus?

Ans: It is an applied predicate calculus specifically tailored for relational databases proposed by E.F. Codd. E.g. of languages based on it are DSL ALPHA, QUEL.

Q.9 How does Tuple-oriented relational calculus differ from domain-oriented relational calculus?

Ans:

1. The **tuple-oriented calculus** uses a tuple variables i.e., variable whose only permitted values are tuples of that relation. E.g. QUEL
2. The **domain-oriented calculus** has domain variables i.e., variables that range over the underlying domains instead of over relation. E.g. ILL, DEDUCE.

Q.10 What is indexing and what are the different kinds of indexing?

Ans: Indexing is a technique for determining how quickly specific data can be found.

Types:

1. Binary search style indexing
2. B-Tree indexing
3. Inverted list indexing
4. Memory resident table
5. Table indexing

Q.11 What is system catalog or catalog relation? How is better known as?

Ans: A RDBMS maintains a description of all the data that it contains, information about every relation and index that it contains. This information is stored in a collection of relations maintained by the system called metadata. It is also called data dictionary.

Q.12 What are the responsibilities of a DBA? If we assume that the DBA is never interested in running his or her own queries, does the DBA still need to Sunderstand query optimization? Why?

Ans: The DBA is responsible for:

Designing the logical and physical schemas, as well as widely-used portions of the external schema.

Security and authorization.

Data availability and recovery from failures.

Database tuning: The DBA is responsible for evolving the database, in particular the conceptual and physical schemas, to ensure adequate performance as user requirements change.

A DBA needs to understand query optimization even if s/he is not interested in running his or her own queries because some of these responsibilities (database design and tuning) are related to query optimization. Unless the DBA understands the performance needs of widely used queries, and how the DBMS will optimize and execute these queries, good design and tuning decisions cannot be made.

Q.13 Define the following terms: *relation schema, relational database schema, domain,* attribute, attribute domain, relation instance, relation cardinality, and relation degree.

Ans: A relation schema can be thought of as the basic information describing a table or relation. This includes a set of column names, the data types associated with each column, and the name associated with the entire table. For example, a
relation schema for the relation called Students could be expressed using the following representation:
Students(*sid:* string, *name:* string, *login:* string, *age:* integer, *gpa:* real)

There are five fields or columns, with names and types as shown above.
A ***relational database schema*** is a collection of relation schemas, describing one or more relations.

Domain is synonymous with *data type*. *Attributes* can be thought of as columns in a table. Therefore, an *attribute domain* refers to the data type associated with a column.

A ***relation instance*** is a set of tuples (also known as *rows* or *records*) that each conform to the schema of the relation.

The ***relation cardinality*** is the number of tuples in the relation.

The ***relation degree*** is the number of fields (or columns) in the relation.

Q.14 What are the unary operations in Relational Algebra?

Ans: PROJECTION and SELECTION.

Chapter 7

Data and Query Processing

Q.1 What is DDL (Data Definition Language)?

Ans: A data base schema is specifies by a set of definitions expressed by a special language called DDL.

Q.2 What is VDL (View Definition Language)?

Ans: It specifies user views and their mappings to the conceptual schema.

Q.3 What is SDL (Storage Definition Language)?

Ans: This language is to specify the internal schema. This language may specify the mapping between two schemas.

Q.4 What is Data Storage - Definition Language?

Ans: The storage structures and access methods used by database system are specified by a set of definition in a special type of DDL called data storage-definition language.

Q.5 What is DML (Data Manipulation Language)?

Ans: This language that enable user to access or manipulate data as organised by appropriate data model.

1. **Procedural DML or Low level:** DML requires a user to specify what data are needed and how to get those data.
2. **Non-Procedural DML or High level:** DML requires a user to specify what data are needed without specifying how to get those data.

Q.6 What is DML Compiler?

Ans: It translates DML statements in a query language into low-level instruction that the query evaluation engine can understand.

Q.7 What is Query evaluation engine?

Ans: It executes low-level instruction generated by compiler.

Q.8 What is DDL Interpreter?

Ans: It interprets DDL statements and record them in tables containing metadata.

Q.9 What is Record-at-a-time?

Ans: The Low level or Procedural DML can specify and retrieve each record from a set of records. This retrieve of a record is said to be Record-at-a-time.

Q.10 What is Set-at-a-time or Set-oriented?

Ans: The High level or Non-procedural DML can specify and retrieve many records in a single DML statement. This retrieve of a record is said to be Set-at-a-time or Set-oriented.

Q.11 What is meant by query optimization?

Ans: The phase that identifies an efficient execution plan for evaluating a query that has the least estimated cost is referred to as query optimization.

Q.12 What do you mean by atomicity and aggregation?

Ans:

1. **Atomicity:** Either all actions are carried out or none are. Users should not have to worry about the effect of incomplete transactions. DBMS ensures this by undoing the actions of incomplete transactions.
2. **Aggregation:** A concept which is used to model a relationship between a collection of entities and relationships. It is used when we need to express a relationship among relationships.

Q.13 What is a Phantom Deadlock?

Ans: In distributed deadlock detection, the delay in propagating local information might cause the deadlock detection algorithms to identify deadlocks that do not really exist. Such situations are called phantom deadlocks and they lead to unnecessary aborts.

Q.14 What is a checkpoint and When does it occur?

Ans: A Checkpoint is like a snapshot of the DBMS state. By taking checkpoints, the DBMS can reduce the amount of work to be done during restart in the event of subsequent crashes.

Q.15 What are the different phases of transaction?

Ans: Different phases are

1.) Analysis phase,
2.) Redo Phase,
3.) Undo phase.

Q.16 Which of the following plays an important role in *representing* information about the real world in a database? Explain briefly.

1. The data definition language.
2. The data manipulation language.
3. The buffer manager.
4. The data model.

Ans: Let us discuss the choices in turn.

The data definition language is important in representing information because it is used to describe external and logical schemas.

The data manipulation language is used to access and update data; it is not important for representing the data. (Of course, the data manipulation language must be aware of how data is represented, and reflects this in the constructs that it supports.)

The buffer manager is not very important for representation because it brings arbitrary disk pages into main memory, independent of any data representation.

The data model is fundamental to representing information. The data model determines what data representation mechanisms are supported by the DBMS. The data definition language is just the specific set of language constructs available to describe an actual application's data in terms of the *data model*.

Q.17 What is SQL? Structured Query Langauge.

Ans: SQL is a globally accepted standard language that is used to handle database related activities for creating database schemas, updating/inserting/deleting of data, retrievieng data, and managing user roles and databases activities.

It stands for **<u>Structured Query Language</u>**

SQL is packaged and marketed by several companies. A few popuplar versions of SQL are Oracle SQL (7,8,9i,10g,11i), Microsoft SQL Server (7,2000,2005,2008), IBM DB2, MySQL, Microsoft Access, Postgres SQL, Sybase etc.

Q.18 Define SQL and state the differences between SQL and other conventional programming Languages.

Ans: SQL is a nonprocedural language that is designed specifically for data access operations on normalized relational database structures. The primary difference between SQL and other conventional programming languages is that SQL statements specify what data operations should be performed rather than how to perform them.

Q.19 What are the characteristics of SQL?

Ans:

(i) SQL enables end user and system persons to deal with a number of database managment systems where it is available.

(ii) Applications written in SQL can be easily ported across systems. Such porting could be required when the underlying DBMS needs to upgraded because of change in transaction volumes or when a system developed in one environment is to be used on another.

(iii) SQL as a language is independent of the way it is implented internally. A query returns the same result regardless of whether optimizing has been done with indexes or not. This is because SQL specifies what is required and not how it is to be done.

(iv) The language while being simple and easy to learn can cope with complex situations.

(v) The results to be expected are well defined in SQL.

Q.20 What is a Cartesian product? What causes it?

Ans: A Cartesian product is the result of an unrestricted join of two or more tables. The result set of a three table Cartesian product will have x * y * z number of rows where x, y, z correspond to the number of rows in each table involved in the join. It is causes by specifying a table in the FROM clause without joining it to another table.

Q.21 What is an advantage to using a stored procedure as opposed to passing an SQL query from an application?

Ans: A stored procedure is pre-loaded in memory for faster execution. It allows the DBMS control of permissions for security purposes. It also eliminates the need to recompile components when minor changes occur to the database.

Q.22 What is a view?

Ans: If we have several tables in a db and we want to view only specific columns from specific tables we can go for views. It would also suffice the needs of security some times allowing specific users to see only specific columns based on the permission that we can configure on the view. Views also reduce the effort that is required for writing queries to access specific columns every time.

Chapter 8

Structured Query Languages

Q.1 Consider the following relations:

Student(*snum:* integer, *sname:* string, *major:* string, *level:* string, *age:* integer)
Class(*name:* string, *meets at:* string, *room:* string, *fid:* integer)
Enrolled(*snum:* integer, *cname:* string)
Faculty(*fid*: integer, *fname:* string, *deptid:* integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or enrolled in a course taught by I. Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.
4. Find the names of all students who are enrolled in two classes that meet at the same time.
5. Find the names of faculty members who teach in every room in which some class is taught.
6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.
7. For each level, print the level and the average age of students for that level.
8. For all levels except JR, print the level and the average age of students for that level.

9. For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught.

10. Find the names of students enrolled in the maximum number of classes.

11. Find the names of students not enrolled in any class.

12. For each age value that appears in Students, find the level value that appears most often. For example, if there are more FR level students aged 18 than SR, JR, or so students aged 18, you should print the pair (18, FR).

Ans: The answers are given below:

```
1. SELECT DISTINCT S.Sname
   FROM Student S, Class C, Enrolled E, Faculty F
   WHERE S.snum = E.snum AND E.cname = C.name AND C.fid = F.fid AND
   F.fname = ‘I.Teach’ AND S.level = ‘JR’

2. SELECT MAX(S.age)
   FROM Student S
   WHERE (S.major = ‘History’)
   OR S.snum IN (SELECT E.snum
   FROM Class C, Enrolled E, Faculty F
   WHERE E.cname = C.name AND C.fid = F.fid
   AND F.fname = ‘I.Teach’ )

3. SELECT C.name
   FROM Class C
   WHERE C.room = ‘R128’
   OR C.name IN (SELECT E.cname
   FROM Enrolled E
   GROUP BY E.cname
   HAVING COUNT (*) >= 5)
```
SQL: Queries, Constraints, Triggers 47

```
4. SELECT DISTINCT S.sname
   FROM Student S
   WHERE S.snum IN (SELECT E1.snum
   FROM Enrolled E1, Enrolled E2, Class C1, Class C2
   WHERE E1.snum = E2.snum AND E1.cname <> E2.cname
   AND E1.cname = C1.name
   AND E2.cname = C2.name AND C1.meets at = C2.meets at)

5. SELECT DISTINCT F.fname
   FROM Faculty F
   WHERE NOT EXISTS (( SELECT *
   FROM Class C )
   EXCEPT
   (SELECTC1.room
   FROM Class C1
   WHERE C1.fid = F.fid ))

 6. SELECT DISTINCT F.fname
   FROM Faculty F
   WHERE 5 > (SELECT COUNT (E.snum)
   FROM Class C, Enrolled E
   WHERE C.name = E.cname
   AND C.fid = F.fid)

7. SELECT S.level, AVG(S.age)
   FROM Student S
   GROUP BY S.level

8. SELECT S.level, AVG(S.age)
   FROM Student S
   WHERE S.level <> 'JR'
   GROUP BY S.level

 9. SELECT F.fname, COUNT(*) AS CourseCount
   FROM Faculty F, Class C
   WHERE F.fid = C.fid
   GROUP BY F.fid, F.fname
   HAVING EVERY ( C.room = 'R128' )

10. SELECT DISTINCT S.sname
   FROM Student S
```

```
WHERE S.snum IN (SELECT E.snum
FROM Enrolled E
GROUP BY E.snum
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Enrolled E2
GROUP BY E2.snum ))
```

11. SELECT DISTINCT S.sname
```
FROM Student S
WHERE S.snum NOT IN (SELECT E.snum
FROM Enrolled E )
```

12. SELECT S.age, S.level
```
FROM Student S
GROUP BY S.age, S.level,
HAVING S.level IN (SELECT S1.level
FROM Student S1
WHERE S1.age = S.age
GROUP BY S1.level, S1.age
HAVING COUNT (*) >= ALL (SELECT COUNT (*)
FROM Student S2
WHERE s1.age = S2.age
GROUP BY S2.level, S2.age))
```

Q.2 The following relations keep track of airline flight information:
Flights(*flno:* integer, *from:* string, *to:* string, *distance:* integer, *departs:* time, *arrives:* time, *price:* real)
Aircraft(*aid:* integer, *aname:* string, *cruisingrange:* integer)
Certified(*eid:* integer, *aid:* integer)
Employees(*eid:* integer, *ename:* string, *salary:* integer)

Note that the Employees relation describes pilots and other kinds of employees as well;
every pilot is certified for some aircraft, and only pilots are certified to fly.

Write each of the following queries in SQL. (*Additional queries using the same schema are listed in the exercises for Chapter 4.*)

1. Find the names of aircraft such that all pilots certified to operate them have salaries more than $80,000.
2. For each pilot who is certified for more than three aircraft, find the *eid* and the maximum *cruising range* of the aircraft for which she or he is certified.
3. Find the names of pilots whose *salary* is less than the price of the cheapest route from Los Angeles to Honolulu.
4. For all aircraft with *cruising range* over 1000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.
5. Find the names of pilots certified for some Boeing aircraft.
6. Find the *aid*s of all aircraft that can be used on routes from Los Angeles to Chicago.
7. Identify the routes that can be piloted by every pilot who makes more than $100,000.
8. Print the *ename*s of pilots who can operate planes with *cruising range* greater than 3000 miles but are not certified on any Boeing aircraft. 50 Chapter 5
9. A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.
10. Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).
11. Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.

12. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles.

13. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircrafts.

14. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles and who are certified on some Boeing aircraft.

Ans: The answers are given below:

```
1. SELECT DISTINCT A.aname
   FROM Aircraft A
   WHERE A.Aid IN (SELECT C.aid
   FROM Certified C, Employees E
   WHERE C.eid = E.eid AND
   NOT EXISTS ( SELECT *
   FROM Employees E1
   WHERE E1.eid = E.eid AND E1.salary < 80000 ))

2. SELECT C.eid, MAX (A.cruisingrange)
   FROM Certified C, Aircraft A
   WHERE C.aid = A.aid
   GROUP BY C.eid
   HAVING COUNT (*) > 3

3. SELECT DISTINCT E.ename
   FROM Employees E
   WHERE E.salary < ( SELECT MIN (F.price)
   FROM Flights F
   WHERE F.from = 'Los Angeles' AND F.to = 'Honolulu' )
```

4. Observe that *aid* is the key for Aircraft, but the question asks for aircraft names;
we deal with this complication by using an intermediate relation Temp:
SQL: Queries, Constraints, Triggers 51
SELECT Temp.name, Temp.AvgSalary

```
   FROM ( SELECT A.aid, A.aname AS name,
   AVG (E.salary) AS AvgSalary
   FROM Aircraft A, Certified C, Employees E
   WHERE A.aid = C.aid AND
   C.eid = E.eid AND A.cruisingrange > 1000
   GROUP BY A.aid, A.aname ) AS Temp

5. SELECT DISTINCT E.ename
   FROM Employees E, Certified C, Aircraft A
   WHERE E.eid = C.eid AND
   C.aid = A.aid AND
   A.aname LIKE 'Boeing%'

6. SELECT A.aid
   FROM Aircraft A
   WHERE A.cruisingrange > ( SELECT MIN (F.distance)
   FROM Flights F
   WHERE F.from = 'Los Angeles' AND F.to = 'Chicago' )

7. SELECT DISTINCT F.from, F.to
   FROM Flights F
   WHERE NOT EXISTS ( SELECT *
   FROM Employees E
   WHERE E.salary > 100000
   AND
   NOT EXISTS (SELECT *
   FROM Aircraft A, Certified C
   WHERE A.cruisingrange > F.distance
   AND E.eid = C.eid
   AND A.aid = C.aid) )

8. SELECT DISTINCT E.ename
   FROM Employees E
   WHERE E.eid IN ( ( SELECT C.eid
   FROM Certified C
   WHERE EXISTS ( SELECT A.aid
   FROM Aircraft A
```

```
  WHERE A.aid = C.aid
  AND A.cruisingrange > 3000 )
  AND
  NOT EXISTS ( SELECT A1.aid
  FROM Aircraft A1
  WHERE A1.aid = C.aid
  AND A1.aname LIKE 'Boeing%' ))

9. SELECT F.departs
  FROM Flights F
  WHERE F.flno IN ( ( SELECT F0.flno
  FROM Flights F0
  WHERE F0.from = 'Madison' AND F0.to = 'New York'
  AND F0.arrives < '18:00' )
  UNION
  ( SELECT F0.flno
  FROM Flights F0, Flights F1
  WHERE F0.from = 'Madison' AND F0.to <> 'New York'
  AND F0.to = F1.from AND F1.to = 'New York'
  AND F1.departs > F0.arrives
  AND F1.arrives < '18:00' )
  UNION
  ( SELECT F0.flno
  FROM Flights F0, Flights F1, Flights F2
  WHERE F0.from = 'Madison'
  AND F0.to = F1.from
  AND F1.to = F2.from
  AND F2.to = 'New York'
  AND F0.to <> 'New York'
  AND F1.to <> 'New York'
  AND F1.departs > F0.arrives
  AND F2.departs > F1.arrives
  AND F2.arrives < '18:00' ))

10. SELECT Temp1.avg - Temp2.avg
  FROM (SELECT AVG (E.salary) AS avg
  FROM Employees E
```

```
WHERE E.eid IN (SELECT DISTINCT C.eid
FROM Certified C )) AS Temp1,
(SELECT AVG (E1.salary) AS avg
FROM Employees E1 ) AS Temp2
```

11. SELECT E.ename, E.salary
```
FROM Employees E
WHERE E.eid NOT IN ( SELECT DISTINCT C.eid
FROM Certified C )
```
SQL: Queries, Constraints, Triggers 53
```
AND E.salary > ( SELECT AVG (E1.salary)
FROM Employees E1
WHERE E1.eid IN
( SELECT DISTINCT C1.eid
FROM Certified C1 ) )
```

12. SELECT E.ename
```
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000)
```

13. SELECT E.ename
```
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000) AND COUNT (*) > 1
```

14. SELECT E.ename
```
FROM Employees E, Certified C, Aircraft A
WHERE C.aid = A.aid AND E.eid = C.eid
GROUP BY E.eid, E.ename
HAVING EVERY (A.cruisingrange > 1000) AND ANY (A.aname = 'Boeing')
```

Q.3 What is the difference between UNION and UNION ALL in SQL?

Ans: UNION is an SQL keyword used to merge the results of two or more tables using a Select statement, containing the same fields, with removed duplicate values. UNION ALL does the same, however it persists duplicate values.

Q.4 What is the need to use a LIKE statement?

Ans: When a partial search is required in a scencario, where for instance, you need to find all employees with the last name having the sequence of characters "gat", then you may use the following query, to match a search criteria:

Select empid, firstname, lastname from t_employee where lastname like '%gats%'

This might search all employees with last name containing the character sequence 'gats' like Gates, Gatsby, Gatsburg, Sogatsky, etc.

% is used to represent remaining all characters in the name. This query fetches all records contains gats in the e middle of the string.

Q.5 Explain the use of the by GROUP BY and the HAVING clause.

Ans: The GROUP BY partitions the selected rows on the distinct values of the column on which the group by has been done. In tandem, the HAVING selects groups which match the criteria specified.

Q.6 What is the difference between IN and BETWEEN, that are used inside a WHERE clause?

Ans: The BETWEEN clause is used to fetch a range of values, whereas the IN clause fetches data from a list of specified values.

]

Q.7 What is a Join in SQL? What are different types of Joins in SQL? Outer Join, Inner Join, Equi Join, Natural Join, Cross Join, Full Outer Join

Ans: **Join** - A join in SQL is a clause that allows merging of records from one or more than one tables in a database. The records from the tables are fetched based on some values that are common to each. See code example below:

Say we have 2 tables, T_EMPLOYEES and T_SALARY

```
Select * from T_EMPLOYEES JOIN T_SALARY
ON
T_EMPLOYEES.EMP_ID = T_SALARY.EMP_ID;
```

Consider the following 2 tables:

T_SHOOTERS

Name	Gun_Type
Ballu Balram	1
Ekgoli Shikari	2
Fauji Bhai	3
Thulla	4
Jackal	4

T_GUNTYPES

Gun_Type	Description
1	Katta Pt. 5
2	Desi Bandook Single Barrel
3	Rocket Launcher
4	Colt

Inner Join - in this type of join, every record in the tables being joined have a matching record. The condition based on which the records are matched is called the **join predicate**.

Implicit Vs. Explicit Inner Joins - When the clause 'Inner Join' is used, it is said to be an explicit join.

See code example below:

```
SELECT * FROM t_shooters
INNER JOIN t_guntypes
ON t_shooters.gun_type = t_guntypes.gun_type;
```

Above is an explicit inner join. The same can be achieved without using the 'inner join' clause like this:

```
SELECT * FROM t_shooter, t_guntype
WHERE t_shooters.gun_type = t_guntypes.gun_type;
```

Above is an implicit inner join.

Equi Join - This is a type of Inner Join. (It is also called a **Theta Join**). - It is a join where the equality '**=**' operator is used. In case an operator like '<' or '>' or any other operator is used, it is not an Equi Join.

An Equi Join may be used by using equality operator or the **Using** clause. See code example below:

```
SELECT T_SHOOTERS.NAME, GUN_TYPE, T_GUNTYPES.DESCRIPTION
FROM T_SHOOTERS INNER JOIN T_GUNTYPES
USING(GUN_TYPE);
```

Natural Join - It is a type of Inner Join. It is a join where in the join predicate is based on all the column names that are common to both the tables being joined. See code example below:

```
SELECT * From T_SHOOTERS NATURAL JOIN T_GUNTYPES;
```

Results will be as below:

Gun_Type	T_Shooters.Name	T_Guntypes.Description
1	Ballu Balram	Katta Pt. 5
2	Ekgoli Shikari	Desi Bandook Single Barrel
3	Fauji Bhai	Rocket Launcher
4	Thulla	Colt
5	Jackal	Colt

Cross Join - Also called **Cartesian Join**. It is the result of joining each row of a table with each row of the other table.

Outer Join - In an Outer join, each record of a table does not really need to match with a record in the corresponding table. Outer joins maybe **Left Outer Joins** or **Right Outher Joins**. Outer Joins are always explicit.

Left Outer Join - This join contains all records from the left table, and matching records in the other table. However, if there are no matching records in the other table, it will still return a result, where in the records of the other table will be NULL.

Right Outer Join - This join fetches all records from the right table and only matching records from the left table. By saying left table, it means the table who's name is to the left of the Join Clause.

See code example below:

```
SELECT * FROM t_shooters LEFT OUTER JOIN t_guntypes
ON t_shooters.gun_type = t_guntypes.gun_type;
```

Full Outer Join - A full outer join merges the result fetched from Left and Right Outer joins. See code example below:

```
SELECT * FROM t_shooters
FULL OUTER JOIN t_guntypes
ON t_shooters.gun_type = t_guntypes.gun_type;
```

Q.8 What are DDL statements in SQL? Explain the Create, Alter and Drop commands?

Ans: In SQL, DDL stands for **Data Definition Language**. It is the part of SQL programming language that deals with the construction and alteration of database structures like tables, views, and further the entities inside these tables like columns. It may be used to set the properties of columns as well.

The three popular commands used in DDL are:
Create - Used to create tables, views, and also used to create functions, stored procedures, triggers, indexes etc.

```
-- An example of Create command below
CREATE TABLE t_students (
stud_id NUMBER(10) PRIMARY KEY,
first_name VARCHAR2(20) NULL,
last_name VARCHAR2(20) NOT NULL,
dateofbirth DATE NULL);
```

Drop - Used to totally eliminate a table, view, index from a database - which means that the records as well as the total structure is eliminated from the database.

```
-- An example of Drop command below
DROP TABLE t_students;
```

Alter - Used to alter or in other words, change the structure of a table, view, index. This is particularly used when there is a scenario wherein the properties of fields inside a table, view, index are supposed to be updated.

```
-- An example of Alter command below
ALTER TABLE t_students ADD address VARCHAR2(200); -- adds a column
ALTER TABLE t_students DROP COLUMN dateofbirth; -- drops a column
ALTER TABLE t_students MODIFY COLUMN address VARCHAR2(100); --
drops a column

-- You can also modify multiple columns using a single modify clause ALTER
TABLE t_students MODIFY
{
COLUMN address VARCHAR2(100)
COLUMN first_name VARCHAR2(50)
COLUMN last_name VARCHAR2(50)
};

-- You can also add constraints like NOT NULL using the Modify statement
ALTER TABLE t_students Modify
{ first_name VARCHAR2(50) NOT NULL };
```

Q.9 What are DML commands in SQL? Select, Insert, Update, Delete?

Ans: **DML** - Stands for **Data Manipulation Language**, its the part of SQL that deals with querying, updating, deleting and inseting records in tables, views.

The following types of actions may be performed using DML commands:

1) **Select** - This command is used to fetch a result set of records from a table, view or a group of tables, views by making use of SQL joins.

Retrieval of data using SQL statements can be done by using different predicates along with it like
Where
Group By
Having
Order By

-- The simplest example of a select statement where in a user wants to
-- retrieve all the records of a table, can be performed by using '*'
-- Use an asterisk character to retrieve all records of a table

SELECT * FROM t_students

The **Where** clause is used with DML statements to check for a condition being met in row.

SELECT * FROM t_students where age > 12 and age < 16;

-- Another way
SELECT * FROM t_students where age between 12 and 16;

SELECT * FROM t_students where name like 'R%'
-- Query above uses the **like** predicate along with a wildcard
-- The result will retrieve all names starting with character 'R'

The **Group By** statement in SQL is used for aggregation, which means that the result that is returned is based on grouping of results based on a column aggregation.

SELECT Roll_No, SUM(Marks) FROM t_students
WHERE Class = 5
GROUP BY Roll_No

The **Having** statement in SQL makes sure that an SQL SELECT statement should only return rows where aggregate values match conditions that are stated.

```
SELECT student_id, SUM(Marks) FROM t_students
WHERE Admission_Date = '01-Apr-2009'
GROUP BY student_id
HAVING SUM(Marks) > 500
```

The **Order By** clause in SQL is used to set the sequence of the output in terms of being alphabetical, magnitude of size, order of date. It may accompanied by an 'asc' or 'desc' clause so as to specify whether the results are in ascending or descending order. Note: The results of a select query that does not use asc or desc is in ascending order, by default.

```
SELECT fname, lname FROM t_students ORDER BY fname ASC;
```

2) **Insert** - This command is used to add record(s) to a table. While inserting a record using the insert statement, the number of records being entered should match the columns of the table. In case the number of items being entered are less than the number of columns, in that case the field names also need to be specified along with the insert statement. See code example below:

Consider a table named t_employees with the following fields:
Emp_Id, FirstName, LastName, Height, Weight
The syntax to insert a record in this table will be:

```
INSERT INTO T_EMPLOYEES VALUES ('445','Amitabh','Bachan','6ft','85kg');
```

What if not all the items need to be insert? Do the following:

```
INSERT INTO T_EMPLOYEES (Emp_Id, FirstName, Height)
VALUES ('445','Amitabh','6ft');
```

Further, an Insert statement can also be used in combination with Select statement. What we can do is that the result of the Select statement may be used as the values to be inserted in a table. See code example below:

```
INSERT INTO T_EMPLOYEES
SELECT * FROM T_SOME_OTHER_EMPLOYEE_TABLE
WHERE FirstName IN ('Pappu', 'Ramu','Kallu','Gabbar');
```

You may even insert specific columns like below:

```
INSERT INTO T_EMPLOYEES (Emp_Id, FirstName)
SELECT Emp_Id, FirstName FROM T_SOME_OTHER_EMPLOYEE_TABLE
WHERE FirstName IN ('Mogambo', 'Dr. Dang','Shakaal','Gabbar','Ajgar
Jurraat','Bhaktawar','Bad Man', 'Prem','Billa Jilani');
```

3) **Update** - This command is used to edit the record(s) of a table. It may be used to update a single row based on a condition, all rows, or a set of rows based on a condition.

It is used along with the **set** clause. Optionally, a **where** clause may be used to match conditions. See code example below:

```
UPDATE TABLE T_EMPLOYEES SET FIRSTNAME = 'Anthony'
WHERE EMP_ID = '445';
```

More examples below:

Update the value of a column

```
UPDATE TABLE T_EMPLOYEES SET AGE = AGE + 1;
```

Update multiple columns in one statement

```
UPDATE TABLE T_SALARY SET
BONUS = BONUS + 10000,
BASIC = BASIC + (0.2 * BONUS);
```

4) **Delete** - This command is used to remove record(s) from a table. All records may be removed in one go, or a set of records may be deleted based on a condition. See code example below:

DELETE FROM T_VILLAINS WHERE FIRSTNAME = 'Pappu';

Code below deletes record(s) based on a condition

DELETE FROM T_VILLAINS WHERE AGE < 18 ;

Delete may also be done based on the result of a sub query:

DELETE FROM T_VILLAINS WHERE AGE IN
(SELECT AGE FROM T_VILLAINS WHERE AGE < 18)

Q.10 What is a database? Write the SQL syntax to create a database .

Ans: A database is a repository of data, the collectively comprises of tables that are used to hold data, views that fetch data from tables, stored procedures, triggers, functionsetc.

A database may comprise a single or multiple tables. An SQL server may comprise of multiple databases, of course this also depends on the SQL Server software being used.

Further, a database may comprise of a single or multiple tables.

The relationships between different tables in a database are often referred to as **Database Schema**.

Say we have a database called 'db_Employees', in order to query this database, the following syntax may be used in MS SQL:

use db_Emloyees -- This syntax is used in order to use this DB.

The question that comes to mind is, **how is the above database created?**

See the syntax below:

create database db_Employees --Syntax to create a database

Q.11 What is the difference between a Table, View and Synonym in SQL?

Ans: A **Table** is a repository of data, where in the table itself is a physical entity. The table resides physically in the database.

A **View** is not a part of the database's physical representation. It is precompiled, so that data retrieval behaves faster, and also provide a secure accessibility mechanism.

See code example below:

```
Create table T_EMPLOYEE
(Emp_Id integer primary key,
Name varchar2(50)
Skillset varchar2(200),
Salary number(12,2),
DOB datetime);
```

Say there is a scenario where Salary is not to be shown to a group of users, a **View** may be created to display allowable information:

```
Create view EMP_SOME_DETAILS
as
(Select Emp_Id, Name, Skillset, DOB
From T_EMPLOYEE);
```

The advantages of using a view are as follows:
- It may access data from a table, multiple tables, view, multiple views, or a combination of these
- A view connects to the data of its base table(s).
- Provides a secure mechanism of data accessibility

Synonym is alternate name assigned to a table, view, sequence or program unit.
- It may be used to shadow the original name and owner of the actual entity
- Extends the reach of tables, by allowing public access to the synonym .

Q.12 Are the resulting relations of PRODUCT and JOIN operation the same?

Ans: No.
PRODUCT: Concatenation of every row in one relation with every row in another.
JOIN: Concatenation of rows from one relation and related rows from another.

Q.13 Name the three major set of files on disk that compose a database in Oracle.

Ans: There are three major sets of files on disk that compose a database. All the files are binary. These are
1.) Database files
2.) Control files
3.) Redo logs
The most important of these are the database files where the actual data resides. The control files and the redo logs support the functioning of the architecture itself. All three sets of files must be present, open, and available to Oracle for any data on the database to be useable. Without these files, you cannot access the database, and the database administrator might have to recover some or all of the database using a backup, if there is one.

Q.14 What is the difference of a LEFT JOIN and an INNER JOIN statement?

Ans: A LEFT JOIN will take ALL values from the first declared table and matching values from the second declared table based on the column the join has been declared on. An INNER JOIN will take only matching values from both tables

Q.15 When a query is sent to the database and an index is not being used, what type of execution is taking place?

Ans: A table scan.

Q.16 What is a Stored Procedure?

Ans: Its nothing but a set of T-SQL statements combined to perform a single task of several tasks. Its basically like a Macro so when you invoke the Stored procedure, you actually run a set of statements.

Q.17 Can you give an example of Stored Procedure?

Ans: sp_helpdb , sp_who2, sp_renamedb are a set of system defined stored procedures. We can also have user defined stored procedures which can be called in similar way.

Q. 18 Explain Operators in SQL.

Ans: SQL Operators Overview

An operator manipulates individual data items and returns a result. The data items are called operands or arguments. Operators are represented by special characters or by keywords. For example, the multiplication operator is represented by an asterisk (*) and the operator that tests for nulls is represented by the keywords IS NULL. There are two general classes of operators: unary and binary. SQL SQL also supports set operators.

Unary Operators

A unary operator uses only one operand. A unary operator typically appears with its operand in the following format:
operator operand

Binary Operators

A binary operator uses two operands. A binary operator appears with its operands in the following format: operand1 operator operand2

Set Operators

Set operators combine sets of rows returned by queries, instead of individual data items. All set operators have equal precedence. SQL supports the following set operators:

UNION
UNION ALL
INTERSECT
MINUS

The following lists the levels of precedence among the SQL SQL operators from high to low. Operators listed on the same line have the same level of precedence:

Levels of Precedence of the SQL SQL Operators Precedence Level SQL Operator

1	Unary + - arithmetic operators, PRIOR operator
2	* / arithmetic operators
3	Binary + - arithmetic operators, || character operators
4	All comparison operators
5	NOT logical operator
6	AND logical operator
7	OR logical operator

Other Operators
Other operators with special formats accept more than two operands. If an operator receives a null operator, the result is always null. The only operator that does not follow this rule is CONCAT.

Arithmetic Operators

Arithmetic operators manipulate numeric operands. The - operator is also used in date arithmetic.

Arithmetic Operators	**Description**	**Example**
+ (unary)	Makes operand positive	SELECT +3 FROM DUAL;
- (unary)	Negates operand	SELECT -4 FROM DUAL;
/	Division (numbers and dates)	SELECT SAL / 10 FROM EMP;
*	Multiplication	SELECT SAL * 5 FROM EMP;
+	Addition (numbers and dates)	SELECT SAL + 200 FROM EMP;
-	Subtraction (numbers and dates)	SELECT SAL - 100 FROM EMP;

Character Operators
Character operators are used in expressions to manipulate character strings.

Character Operators Operator	Description	Example
||	Concatenates character strings	SELECT 'The Name of the employee is: ' || ENAME FROM EMP;

Concatenating Character Strings

You can concatenate character strings with the following results:
Concatenating two character strings results in another character string.
SQL preserves trailing blanks in character strings by concatenation, regardless of the strings' datatypes.
SQL provides the CONCAT character function as an alternative to the vertical bar operator. For example:

SELECT CONCAT (CONCAT (ENAME, ' is a '),job) FROM EMP WHERE SAL > 2000;

This returns:

```
CONCAT(CONCAT(ENAME
-------------------------
KING      is a PRESIDENT
BLAKE     is a MANAGER
CLARK     is a MANAGER
JONES     is a MANAGER
FORD      is a ANALYST
SCOTT     is a ANALYST

6 rows selected.
```

SQL treats zero-length character strings as nulls. When you concatenate a zero-length character string with another operand the result is always the other operand. A null value can only result from the concatenation of two null strings.

Comparison Operators
Comparison operators are used in conditions that compare one expression with another. The result of a comparison can be TRUE, FALSE, or UNKNOWN.

Comparison Operators Operator	Description	Example
=	Equality test.	SELECT ENAME "Employee" FROM EMP WHERE SAL = 1500;
!=, ^=, <>	Inequality test.	SELECT ENAME FROM EMP WHERE SAL ^= 5000;
>	Greater than test.	SELECT ENAME "Employee", JOB "Title" FROM EMP WHERE SAL > 3000;
<	Less than test.	SELECT * FROM PRICE WHERE MINPRICE < 30;
>=	Greater than or equal to test.	SELECT * FROM PRICE WHERE MINPRICE >= 20;
<=	Less than or equal to test.	SELECT ENAME FROM EMP WHERE SAL <= 1500;
IN	"Equivalent to any member of" test. Equivalent to "= ANY".	SELECT * FROM EMP WHERE ENAME IN ('SMITH', 'WARD');
ANY/ SOME	Compares a value to each value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, or >=. Evaluates to FALSE if the query returns no rows.	SELECT * FROM DEPT WHERE LOC = SOME ('NEW YORK','DALLAS');
NOT IN	Equivalent to "!= ANY". Evaluates to FALSE if any member of the set is NULL.	SELECT * FROM DEPT WHERE LOC NOT IN ('NEW YORK', 'DALLAS');
ALL	Compares a value with every value in a list or returned by a query. Must be preceded by =, !=, >, <, <=, or >=. Evaluates	SELECT * FROM emp WHERE sal >= ALL (1400, 3000);

	to TRUE if the query returns no rows.	
[NOT] BETWEEN x and y	[Not] greater than or equal to x and less than or equal to y.	SELECT ENAME, JOB FROM EMP WHERE SAL BETWEEN 3000 AND 5000;
EXISTS	TRUE if a sub-query returns at least one row.	SELECT * FROM EMP WHERE EXISTS (SELECT ENAME FROM EMP WHERE MGR IS NULL);
x [NOT] LIKE y [ESCAPE z]	TRUE if x does [not] match the pattern y. Within y, the character "%" matches any string of zero or more characters except null. The character "_" matches any single character. Any character following ESCAPE is interpretted litteraly, useful when y contains a percent (%) or underscore (_).	SELECT * FROM EMP WHERE ENAME LIKE '%E%';
IS [NOT] NULL	Tests for nulls. This is the only operator that should be used to test for nulls.	SELECT * FROM EMP WHERE COMM IS NOT NULL AND SAL > 1500;
Comparison Operators Operator	Description	Example

Logical Operators

Logical operators manipulate the results of conditions.

Logical Operators Operator	Description	Example

NOT	Returns TRUE if the following condition is FALSE. Returns FALSE if it is TRUE. If it is UNKNOWN, it remains UNKNOWN.	SELECT * FROM EMP WHERE NOT (job IS NULL)
		SELECT * FROM EMP WHERE NOT (sal BETWEEN 1000 AND 2000)

AND	Returns TRUE if both component conditions are TRUE. Returns FALSE if either is FALSE; otherwise returns UNKNOWN.	SELECT * FROM EMP WHERE job='CLERK' AND deptno=10
OR	Returns TRUE if either component condition is TRUE. Returns FALSE if both are FALSE. Otherwise, returns UNKNOWN.	SELECT * FROM emp WHERE job='CLERK' OR deptno=10

Set Operators
Set operators combine the results of two queries into a single result.

Set Operators Operator	**Description**	**Example**

UNION	Returns all distinct rows selected by either query.	SELECT ENAME FROM EMP WHERE JOB = 'CLERK'
		
		UNION
		
		SELECT ENAME FROM EMP WHERE JOB = 'ANALYST';

UNION ALL	Returns all rows selected by either query, including all duplicates.	(SELECT SAL FROM EMP WHERE JOB = 'CLERK' UNION ALL SELECT SAL FROM EMP WHERE JOB = 'ANALYST');
INTERSECT and INTERSECT ALL	Returns all distinct rows selected by both queries.	SELECT * FROM orders_list1 INTERSECT

		SELECT * FROM orders_list2

MINUS	Returns all distinct rows selected by the first query but not the second.	SELECT * FROM (SELECT SAL FROM EMP WHERE JOB = 'PRESIDENT'
		
		MINUS
		
		SELECT SAL FROM EMP WHERE JOB = 'MANAGER');

Other Operators
The following lists other operators:

Other Operators	**Description**	**Example**
(+)	Indicates that the preceding column is the outer join column in a join.	SELECT ENAME, DNAME FROM EMP, DEPT WHERE DEPT.DEPTNO = EMP.DEPTNO (+);
PRIOR	Evaluates the following expression for the parent row of the current row in a hierarchical, or tree-structured query. In such a query, you must use this operator in the CONNECT BY clause to define the relationship between the parent and child rows.	SELECT EMPNO, ENAME, MGR FROM EMP CONNECT BY PRIOR EMPNO = MGR;

Chapter 9

Advanced Features of SQL

Q.1 Explain the term *stored procedure*, and give examples why stored procedures are useful.

Ans: **Stored procedures** are programs that run on the database server and can be called with a single SQL statement. They are useful in situations where the processing should be done on the server side rather than the client side. Also, since the procedures are centralized to the server, code writing and maintenance is simplified, because the client programs do not have to duplicate the application logic. Stored procedures can also be used to reduce network communication; the results of a stored procedure can be analyzed and kept on the database server.

Q.2 Define the "integrity rules"?

Ans: There are two Integrity rules.

1. **Entity Integrity:** States that "Primary key cannot have NULL value"
2. **Referential Integrity:** States that "Foreign Key can be either a NULL value or should be Primary Key value of other relation.

Q.3 What is extension and intension?

Ans:

1. **Extension:** It is the number of tuples present in a table at any instance. This is time dependent.
2. **Intension:** It is a constant value that gives the name, structure of table and the constraints laid on it.

Q.4 Name the sub-systems of a RDBMS.

Ans: I/O, Security, Language Processing, Process Control, Storage Management, Logging and Recovery, Distribution Control, Transaction Control, Memory Management, Lock Management.

Q.5 Which part of the RDBMS takes care of the data dictionary? How?

Ans: Data dictionary is a set of tables and database objects that is stored in a special area of the database and maintained exclusively by the kernel.

Q.6 What is the job of the information stored in data-dictionary?

Ans: The information in the data dictionary validates the existence of the objects, provides access to them, and maps the actual physical storage location.

Q.7 How do you communicate with an RDBMS?

Ans: You communicate with an RDBMS using Structured Query Language (SQL).

Q.8 What is database Trigger?

Ans: A database trigger is a PL/SQL block that can defined to automatically execute for insert, update, and delete statements against a table. The trigger can e defined to execute once for the entire statement or once for every row that is inserted, updated, or deleted. For any one table, there are twelve events for which you can define database triggers. A database trigger can call database procedures that are also written in PL/SQL.

Q.9 What are stored-procedures? And what are the advantages of using them?

Ans: Stored procedures are database objects that perform a user defined operation. A stored procedure can have a set of compound SQL statements. A stored procedure executes the SQL commands and returns the result to the client. Stored procedures are used to reduce network traffic.

Q.10 What is Storage Manager?

Ans: It is a program module that provides the interface between the low-level data stored in database, application programs and queries submitted to the system.

Q.11 What is Buffer Manager?

Ans: It is a program module, which is responsible for fetching data from disk storage into main memory and deciding what data to be cache in memory.

Q.12 What is Transaction Manager?

Ans: It is a program module, which ensures that database, remains in a consistent state despite system failures and concurrent transaction execution proceeds without conflicting.

Q.13 What is File Manager?

Ans: It is a program module, which manages the allocation of space on disk storage and data structure used to represent information stored on a disk.

Q.14 What is Authorization and Integrity manager?

Ans: It is the program module, which tests for the satisfaction of integrity constraint and checks the authority of user to access data.

Q.15 What are cursors give different types of cursors?

Ans: PL/SQL uses cursors for all database information accesses statements. The language supports the use two types of cursors

1.) Implicit

2.) Explicit

Q.16 What is cold backup and hot backup (in case of Oracle)?

Ans:

1. **Cold Backup:** It is copying the three sets of files (database files, redo logs, and control file) when the instance is shut down. This is a straight file copy, usually from the disk directly to tape. You must shut down the instance to guarantee a consistent copy. If a cold backup is performed, the only option available in the event of data file loss is restoring all the files from the latest backup. All work performed on the database since the last backup is lost.
2. **Hot Backup:** Some sites (such as worldwide airline reservations systems) cannot shut down the database while making a backup copy of the files. The cold backup is not an available option.

Q.17 What are the pros and cons of using triggers?

Ans: A trigger is one or more statements of SQL that are being executed in event of data modification in a table to which the trigger belongs.

Triggers enhance the security, efficiency, and standardization of databases.

Triggers can be beneficial when used:

- to check or modify values before they are actually updated or inserted in the database. This is useful if you need to transform data from the way the user sees it to some internal database format.

- to run other non-database operations coded in user-defined functions

- to update data in other tables. This is useful for maintaining relationships between data or in keeping audit trail information.

- to check against other data in the table or in other tables. This is useful to ensure data integrity when referential integrity constraints aren't appropriate, or when table check constraints limit checking to the current table only.

Q.18 What is a trigger?
Ans: Triggers are basically used to implement business rules. Triggers is also similar to stored procedures. The difference is that it can be activated when data is added or edited or deleted from a table in a database.

Q.19 What is an Index?
Ans: When queries are run against a db, an index on that db basically helps in the way the data is sorted to process the query for faster and data retrievals are much faster when we have an index.

Q.20 What are the types of indexes available with SQL Server?

Ans: There are basically two types of indexes that we use with the SQL Server. Clustered and the Non-Clustered.

Q.21 What is the basic difference between clustered and a non-clustered index?
Ans: The difference is that, Clustered index is unique for any given table and we can have only one clustered index on a table. The leaf level of a clustered index is the actual data and the data is resorted in case of clustered index. Whereas in case of non-clustered index the leaf level is actually a pointer to the data in rows so we can have as many non-clustered indexes as we can on the db.

Q.22 What are cursors?

Ans: Well cursors help us to do an operation on a set of data that we retreive by commands such as Select columns from table. For example : If we have duplicate records in a table we can remove it by declaring a cursor which would check the records during retreival one by one and remove rows which have duplicate values.

Q.23 What is a transaction and what are ACID properties?

Ans: A transaction is a logical unit of work in which, all the steps must be performed or none. ACID stands for Atomicity, Consistency, Isolation, Durability. These are the properties of a transaction. For more information and explanation of these properties, see SQL Server books online or any RDBMS fundamentals text book. Explain different isolation levels An isolation level determines the degree of isolation of data between concurrent transactions. The default SQL Server isolation level is Read Committed. Here are the other isolation levels (in the ascending order of isolation): Read Uncommitted, Read Committed, Repeatable Read, Serializable. See SQL Server books online for an explanation of the isolation levels. Be sure to read about SET TRANSACTION ISOLATION LEVEL, which lets you customize the isolation level at the connection level.

Read Committed - A transaction operating at the Read Committed level cannot see changes made by other transactions until those transactions are committed. At this level of isolation, dirty reads are not possible but nonrepeatable reads and phantoms are possible. Read Uncommitted - A transaction operating at the Read Uncommitted level can see uncommitted changes made by other transactions. At this level of isolation, dirty reads, nonrepeatable reads, and phantoms are all possible. Repeatable Read - A transaction operating at the Repeatable Read level is guaranteed not to see any changes made by other transactions in values it has already read. At this level of isolation, dirty reads and nonrepeatable reads are not possible but phantoms are possible.

Serializable - A transaction operating at the Serializable level guarantees that all concurrent transactions interact only in ways that produce the same effect as if each transaction were entirely executed one after the other. At this isolation level, dirty reads, nonrepeatable reads, and phantoms are not possible.

Q.24 What is an index? What are the types of indexes? How many clustered indexes can be created on a table? I create a separate index on each column of a table. What are the advantages and disadvantages of this approach?

Ans: Indexes in SQL Server are similar to the indexes in books. They help SQL Server retrieve the data quicker. Indexes are of two types. Clustered indexes and non-clustered indexes. When you create a clustered index on a table, all the rows in the table are stored in the order of the clustered index key. So, there can be only one clustered index per table. Non-clustered indexes have their own storage separate from the table data storage. Non-clustered indexes are stored as B-tree structures (so do clustered indexes), with the leaf level nodes having the index key and it's row locater. The row located could be the RID or the Clustered index key, depending up on the absence or presence of clustered index on the table. If you create an index on each column of a table, it improves the query performance, as the query optimizer can choose from all the existing indexes to come up with an efficient execution plan. At the same t ime, data modification operations (such as INSERT, UPDATE, DELETE) will become slow, as every time data changes in the table, all the indexes need to be updated. Another disadvantage is that, indexes need disk space, the more indexes you have, more disk space is used.

Q.25 Whats the difference between a Stored Procedure and a Function in SQL?

Ans: Primarily, the main difference between a Stored Procedure and a Function in SQL is that a Function has the ability to return a value to the calling routine, where as a Stored Procedure may or may not return a value, depending on how it is programmed. A Function should have a minimum of one return type.

This means that while creating a Function in SQL, it should have a return type.

Also, the results of a function may be assigned to a Select statement; this isn't possible with Stored Procs.

Q.26 What is a Stored Procedure in SQL? Write the syntax for a Stored Proc.

Ans: A **Stored Procedure** is a set of instructions saved for invoking repeated execution. A procedure may be used to handle a set of business logic to be performed. Say we want to insert a record in a table, this type of a scenario is repeatitive and hence a stored procedure may take care of this requirement. Further, the stored procedure may be passed parameters by the calling routine, and so it may be reusable. Below is the syntax to create a procedure:

```
Create or Replace Procedure [SCHEMA.]PROCEDURENAME
([Argument][in|out|in out])
[is|as]
begin
[Statements ....]
end;
```

Example:

```
Create or Replace Procedure usp_Insert_Employee
(fname varchar2(20) in, lname varchar2(20) in)
as
begin
Insert into t_Employee Values
(fname, lname);
end;
```

Q.27 Write the syntax to create a Function in SQL

Ans: Below is the syntax to create a Function in SQL:

```
Create [or Replace] Function [Schema.]FunctionName
(argument [in|out] datatype])
return datatype
is
[variables if any]
begin
[Statements...];
return [return value];
end;
```

See example below:

```
Create or Replace IsEligibleToVote(InputAge in number )
return varchar2
is
 v_name t_citizen.varchar2%type;
 v_age t_citizen.varchar2%type;
 v_eligible varchar2 = 'no';
begin
  if(InputAge > 18) then
   v_name = &&inp1;
    v_age = &&inp2;
     insert into t_citizen values (v_name, v_age) ;
 v_eligible='yes';
end if;
return v_eligible;
end;
```

Q.28 What are views?

Ans A view is like a window through which you can view or change information in a table.a view is a virtual table that is-

It look like a table but it does not exists as such

Its data are derived from base table or tables

It only stores its definition.

A view hide some information of a table or combine information from several table as if information existed in a single table.It is a subset of the information within a table.

Advantages: Convinience,simplicity,security.

Syntax:

```
CREATE OR REPLACE VIEW view_name AS
 SELECT column_name(s)
 FROM table_name
 WHERE condition
```

Example:
Selects every product in the "Products" table with a unit price higher than the average unit price:

```
 CREATE VIEW [Products Above Average Price] AS
 SELECT ProductName,UnitPrice
 FROM Products
 WHERE UnitPrice>(SELECT AVG(UnitPrice) FROM Products)
```

Q.29 What is sequence?

Ans A sequence is a special database object that generate integers according to a specified rules at the time the sequence was created. Sequence have many purposes in database systems, the most common of which is to generate primary keys automatically.

Syntax:

```
CREATE SEQUENCE name
  [ INCREMENT increment_value ]
  [ MINVALUE minimum_value ]
  [ MAXVALUE maximum_value ]
  [ START start_value ]
  [ CACHE cache_value ]
  [ CYCLE ]
```

Variable Description

INCREMENT BY
The increment value. This can be a positive or negative number.

START WITH
The start value for the sequence.

MAXVALUE
The maximum value that the sequence can generate. If specifying NOMAXVALUE, the maximum value is 263-1.

MINVALUE
The minimum value that the sequence can generate. If specifying NOMINVALUE, the minimum value is -263.

CYCLE
Specify CYCLE to indicate that when the maximum value is reached the sequence starts over again at the start value. Specify NOCYCLE to generate an error upon reaching the maximum value.

Using a Sequence

Use sequences when an application requires a unique identifier. INSERT statements, and occasionally UPDATE statements, are the most common places to use sequences. Two "functions" are available on sequences:

NEXTVAL: Returns the next value from the sequence.

CURVAL: Returns the value from the last call to NEXTVAL by the current user during the current connection. For example, if User A calls NEXTVAL and it returns 124, and User B immediately calls NEXTVAL getting 125, User A will get 124 when calling CURVAL, while User B will get 125 while calling CURVAL. It is important to understand the connection between the sequence value and a particular connection to the database. The user cannot call CURVAL until making a call to NEXTVAL at least once on the connection. CURVAL returns the current value returned from the sequence on the current connection, not the current value of the sequence.

Examples

To create the sequence:
```
CREATE SEQUENCE customer_seq INCREMENT BY 1 START WITH 100
```

To use the sequence to enter a record into the database:
```
INSERT INTO customer (cust_num, name, address)
 VALUES (customer_seq.NEXTVAL, 'John Doe', '123 Main St.')
```

To find the value just entered into the database:
SELECT customer_seq.CURVAL AS LAST_CUST_NUM

Q.30 Explain index.

Ans An index is a sorted list of data from one or more columns in the table that are commonly used as selection criteria.
Index are logically and physically independent of the data in the associated table i.e. you can create or drop on index at any time without effecting the base table or other indexes.
Indexes are optional structures associated with tables. You can create indexes to speed SQL statements execution on a table. Indexes are primary means of reducing this input-output when properly used.

SQL CREATE INDEX Syntax:

Creates an index on a table. Duplicate values are allowed:

```
 CREATE INDEX index_name
 ON table_name (column_name)
```

SQL CREATE UNIQUE INDEX Syntax

Creates a unique index on a table. Duplicate values are not allowed:

```
 CREATE UNIQUE INDEX index_name
 ON table_name (column_name)
```

CREATE INDEX Example:

The SQL statement below creates an index named "PIndex" on the "LastName" column in the "Persons" table:

```
CREATE INDEX PIndex
 ON Persons (LastName)
```

If you want to create an index on a combination of columns, you can list the column names within the parentheses, separated by commas:

```
CREATE INDEX PIndex
ON Persons (LastName, FirstName)
```