INDEX

SYLLABUS

Unit–I

Introduction to Databases and Transactions

What is database system, purpose of database system, view of data, relational databases, database architecture, transaction management,

Unit-II

Data Models

The importance of data models, Basic building blocks, Business rules, The evolution of data models, Degrees of data abstraction.

Unit-III

Database Design ,ER-Diagram and Unified Modeling Language

Database design and ER Model: overview, ER-Model, Constraints, ER-Diagrams, ERD

Issues, weak entity sets, Codd's rules, Relational Schemas, Introduction to UML

Relational database model:

Logical view of data, keys, integrity rules.

Relational Database design: features of good relational database design, atomic domain and Normalization (1NF, 2NF, 3NF, BCNF).

Unit-IV

Relational Algebra and Calculus

Relational algebra: introduction, Selection and projection, set operations, renaming, Joins, Division, syntax, semantics. Operators, grouping and ungrouping,

Relational comparison.

Calculus: Tuple relational calculus, Domain relational Calculus, calculus vs algebra, computational capabilities.

Unit-V

Constraints, Views and SQL

What is constraints, types of constrains, Integrity constraints,

Views: Introduction to views, data independence, security, updates on views, comparison between tables and views

SQL: data definition, aggregate function, Null Values, nested sub queries, Joined relations. Triggers.

Unit-VI

Transaction management and Concurrency control

Transaction management: ACID properties, serializability and concurrency control,

Lock based concurrency control (2PL, Deadlocks),Time stamping methods, optimistic methods, database recovery management.

CHAPTER I: INTRODUCTION TO DATABASES AND TRANSACTIONS

Topic Covered:

Introduction to Databases and Transactions

What is database system, purpose of database system, view of data, relational databases, database architecture, transaction management.

Data: It can be defined as the raw information which is processed by computer.

Database: It is nothing but collection of different tables which contains data.

Database Management System (DBMS):

A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data.

Q. What is Database Management System (DBMS)?

- A database-management system (DBMS) is a collection of interrelated data and a Set of programs to access those data. The collection of data, usually referred to as the Database, contains information relevant to an enterprise.
  - The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.
  - Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the information stored, despite system crashes or attempts at unauthorized access.
  - If data are to be shared among several users, the system must avoid possible anomalous results.
  - A database is a computer generated software program which can be used to access the data stored in database in an organized manner.
  - The term database is a structured collection of data stored which can be stored in digital form. Before the actual data is stored in the database, we should clearly specify the schema of the database and different techniques used to manipulate the data stored in a database.
  - Database shouldn't only care about the insertion and modification of the data in the database. At times, it should also focus on how to protect the data stored in the database from unauthorized access.
  - DBMS must provide efficient techniques in order to protect the data from accidental system crashes

---

Q. What are the Applications of Database Management System?

- **Banking**: For customer information, accounts, loans, and banking transactions.
- **Airlines**: For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
- **Universities**: For student information/ course registrations, and grades.
- **Credit card transactions**: For purchases on credit cards and generation of monthly statements.
- **Telecommunication**: for keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
- **Finance**: for storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable on-line trading by customers and automated trading by the firm.
- **Sales**: For customer, product, and purchase information.
- **On-line retailers**: For sales data noted above plus on-line order tracking/ generation of recommendation lists, and maintenance of on-line product evaluations.

- **Manufacturing**: For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
- **Human resources**: For information about employees, salaries, payroll ,taxes, benefits, and for generation of paychecks.

Q. Explain purpose of Database Management Systems.

OR

Q. What are the Drawbacks of File Processing System?

- **Data redundancy and inconsistency**:
  - Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages.
  - Therefore, the same information may be duplicated in several places (files).
  - This redundancy leads to higher storage and access cost.
  - In addition, it may lead to **data inconsistency**; that is, the various copies of the same data may no longer agree.
- **Difficulty in accessing data**:
  - The conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.
  - For Example, consider that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. The clerk asks the data-processing department to generate such a list.
  - Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of *all* students.
  - The university clerk has now two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory.
  - Therefore extraction of the required data is difficult.
- **Data isolation**:
  - Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
- **Integrity problems**:
  - The data values stored in the database must satisfy certain types of **consistency constraints**.
  - Suppose the university maintains an account for each department, and records the balance amount in each account.
  - Suppose also that the university requires that the account balance of a department may never fall below zero.
  - Developers enforce these constraints in the system by adding appropriate code in the various application programs.
  - However, when new constraints are added, it is difficult to change the programs to enforce them.
  - The problem is compounded when constraints involve several data items from different files.
- **Atomicity problems**:
  - A computer system, like any other device, is subject to failure.
  - In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.
  - It is difficult to ensure atomicity in a conventional file-processing system.
- **Concurrent-access anomalies**:
  - To increase the overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously.
  - In such an environment, interaction of concurrent updates is possible and may result in inconsistent data.

- **Security problems**:
  - Not every user of the database system should be able to access all the data.
  - since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

Q. What are the advantages of Database Management System?

The different advantages of DBMS are as follows.

1. **Improved data sharing.**

The DBMS helps create an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.

2. Improved data security.

The more users access the data, the greater the risks of data security breaches. Corporations invest considerable amounts of time, effort, and money to ensure that corporate data are used properly. A DBMS provides a framework for better enforcement of data privacy and security policies.

3. Better data integration.

Wider access to well-managed data promotes an integrated view of the organization's operations and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.

4. Minimized data inconsistency.

Data inconsistency exists when different versions of the same data appear in different places. The probability of data inconsistency is greatly reduced in a properly designed database.

5. Improved data access.

The DBMS makes it possible to produce quick answers to ad hoc queries. From a database perspective, a query is a specific request issued to the DBMS for data manipulation—for example, to read or update the data. Simply put, a query is a question, and an ad hoc query is a spur-of-the-moment question. The DBMS sends back an answer (called the query result set) to the application.

6. Improved decision making.

Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. The quality of the information generated depends on the quality of the underlying data. Data quality is a comprehensive approach to promoting the accuracy, validity, and timeliness of the data. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives.

7. Increased end-user productivity.

The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

Q. What are the Disadvantages of Database Management System?

1. Increased costs.

Database systems require sophisticated hardware and software and highly skilled personnel. The cost of maintaining the hardware, software, and personnel required to operate and manage a database system can be substantial. Training, licensing, and regulation compliance costs are often overlooked when database systems are implemented.

2. Management complexity.

Database systems interface with many different technologies and have a significant impact on a company's resources and culture. The changes introduced by the adoption of a database system must be properly managed to ensure that they help advance the company's objectives. Given the fact that database systems hold crucial company data that are accessed from multiple sources, security issues must be assessed constantly.

3. Maintaining currency.

To maximize the efficiency of the database system, you must keep your system current. Therefore, you must perform frequent updates and apply the latest patches and security measures to all components. Because database technology advances rapidly, personnel training costs tend to be significant. Vendor dependence. Given the heavy investment in technology and personnel training,

companies might be reluctant to change database vendors. As a consequence, vendors are less likely to offer pricing point advantages to existing customers, and those customers might be limited in their choice of database system components.

4. Frequent upgrade/replacement cycles.

DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, but it also costs money to train database users and administrators to properly use and manage the new features.

Q. Differentiate between DBMS and File System.

	File processing System		DBMS
	High rate of redundancy of data exist in a typical file processing system.	1	Redundancy is reduced.
	There is inconsistency of data.	2	Inconsistency of data is reduced.
	No provision for data security.	3	Provision for data security is made.
	No standard representation of data.	4	Standard representation of data is achieved using relational data model.
	Data integrity is not there.	5	Data integrity is there.
	Data cannot be accessed easily.	6	Data can be accessed easily through table structure (row and columns).
	Data cannot be shared.	7	Data can be shared.
	Retrieval of data is time consuming	8	Retrieval of data is easy.

View of Data

A major purpose of a database system is to provide users with an *abstract* view of the data.

- **Data Abstraction**

Many database-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users interactions with the system:

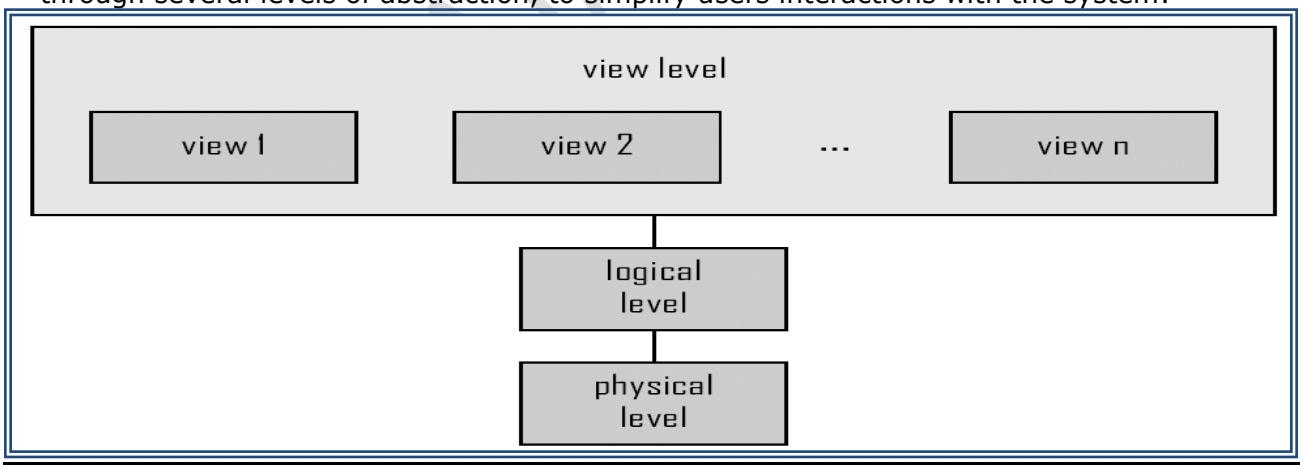


Figure 1.1 The three levels of data abstraction.

- **Physical level**:
  - The lowest level of abstraction describes *how* the data are actually stored.
  - The physical level describes complex low-level data structures in detail.
- **Logical level**:
  - The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data.

- The logical level thus describes the entire database in terms of a small number of relatively simple structures.
- Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. This is referred to as **physical data independence**.
- Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

• **View level**.

- The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database.
- Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Instances and Schemas

Similar to types and variables in programming languages

- **Schema** –

The overall design of the database is called the database **schema.**

The logical structure of the database.

- Example: The database consists of information about a set of customers and accounts and the relationship between them)
- Analogous to type information of a variable in a program
- Physical schema: database design at the physical level
- Logical schema: database design at the logical level

- **Instance** –

The collection of information stored in the database at a particular moment is called an **instance** of the database.

The actual content of the database at a particular point in time

- Analogous to the value of a variable

Data Independence

Data independence can be classified into two types

1.Logical data independence

- It is the ability to modify the conceptual schema without affecting the existing external schemas.
- In logical data independence, the users are shielded from changes in the logical structure of the data or changes in the choice of relations to be stored.
- The changes to the conceptual schema, such as the addition and deletion of entities, addition and deletion of attributes, or addition and deletion of relationships must be possible without changing existing external schemas or having to rewrite application programs.
- Only the view definition and the mapping need be changed in a DBMS that supports logical data independence.

2.Physical data independence

- The ability to modify the internal schema without having to change the conceptual or external schemas is called physical data independence.
- In physical data independence, the conceptual schema insulates the users from changes in the physical storage of the data.
- The changes to the internal schema, such as using different file organizations or storage structures, using different storage devices , modifying indexes or hashing algorithms
- must be possible without changing the conceptual or external schemas.
- In other words, physical data independence indicates that the physical storage structures or devices used for storing the data could be changed without necessitating a change in the conceptual view or any of the external views.

- **Note:**

The Logical data independence is difficult to achieve than physical data independence as it requires the flexibility in the design of database and programmer has to anticipate the future requirements or modifications in the design of the database.

Database Languages

- A database system provides a **data-definition language** to specify the database
- schema and a **data-manipulation language** to express database queries and updates.
- In practice, the data-definition and data-manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

➢ **Data-Manipulation Language**

A **data-manipulation language (DML)** is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- Retrieval of information stored in the database
- Insertion of new information into the database
- Deletion of information from the database
- Modification of information stored in the database

There are basically two types:

- **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
- **Declarative DMLs** (also referred to as **nonprocedural DMLs**) require a user to specify *what* data are needed *without* specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs.

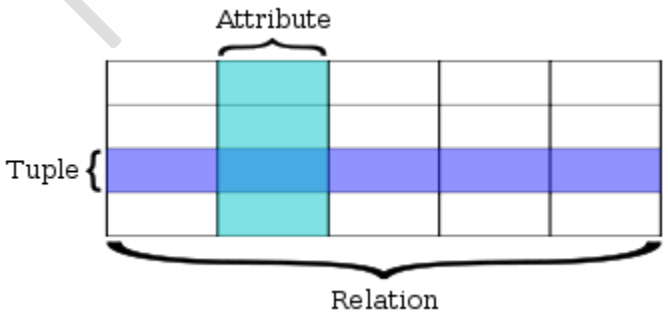
A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**.

➢ **Data-Definition Language**

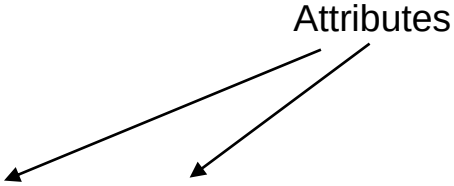
- We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**). The DDL is also used to specify additional properties of the data.
- We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition** language. These statements define the implementation details of the database schemas, which are usually hidden from the users.
- The data values stored in the database must satisfy certain **consistency constraints**.
- For example, suppose the university requires that the account balance of a department must never be negative.
- The DDL provides facilities to specify such constraints. The database system checks these constraints every time the database is updated.

Relational Databases

- A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a DML and DDL.



Relational database terminology.



customer_id	customer_name	customer_street	customer_city	account_number
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-101
192-83-7465	Johnson	12 Alma St.	Palo Alto	A-201
677-89-9011	Hayes	3 Main St.	Harrison	A-102
182-73-6091	Turner	123 Putnam St.	Stamford	A-305
321-12-3123	Jones	100 Main St.	Harrison	A-217
336-66-9999	Lindsay	175 Park Ave.	Pittsfield	A-222
019-28-3746	Smith	72 North St.	Rye	A-201

Example of tabular data in the relational model

Advantages:

1. **Ease of use**: The revision of any information as tables consisting of rows and columns is much easier to understand.
2. **Flexibility**: Different tables from which information has to be linked and extracted can be easily manipulated by operators such as project and join to give information in the form in which it is desired.

3. **Precision**: The usage of relational algebra and relational calculus in the manipulation of the relations between the tables ensures that there is no ambiguity, which may otherwise arise in establishing the linkages in a complicated network type database.

4. **Security**: Security control and authorization can also be implemented more easily by moving sensitive attributes in a given table into a separate relation with its own authorization controls. If authorization requirement permits, a particular attribute could be joined back with others to enable full information retrieval.
5. **Data Independence**: Data independence is achieved more easily with normalization structure used in a relational database than in the more complicated tree or network structure.
6. **Data Manipulation Language**: The possibility of responding to query by means of a language based on relational algebra and relational calculus e.g SQL is easy in the relational database approach. For data organized in other structure the query language either becomes complex or extremely limited in its capabilities.

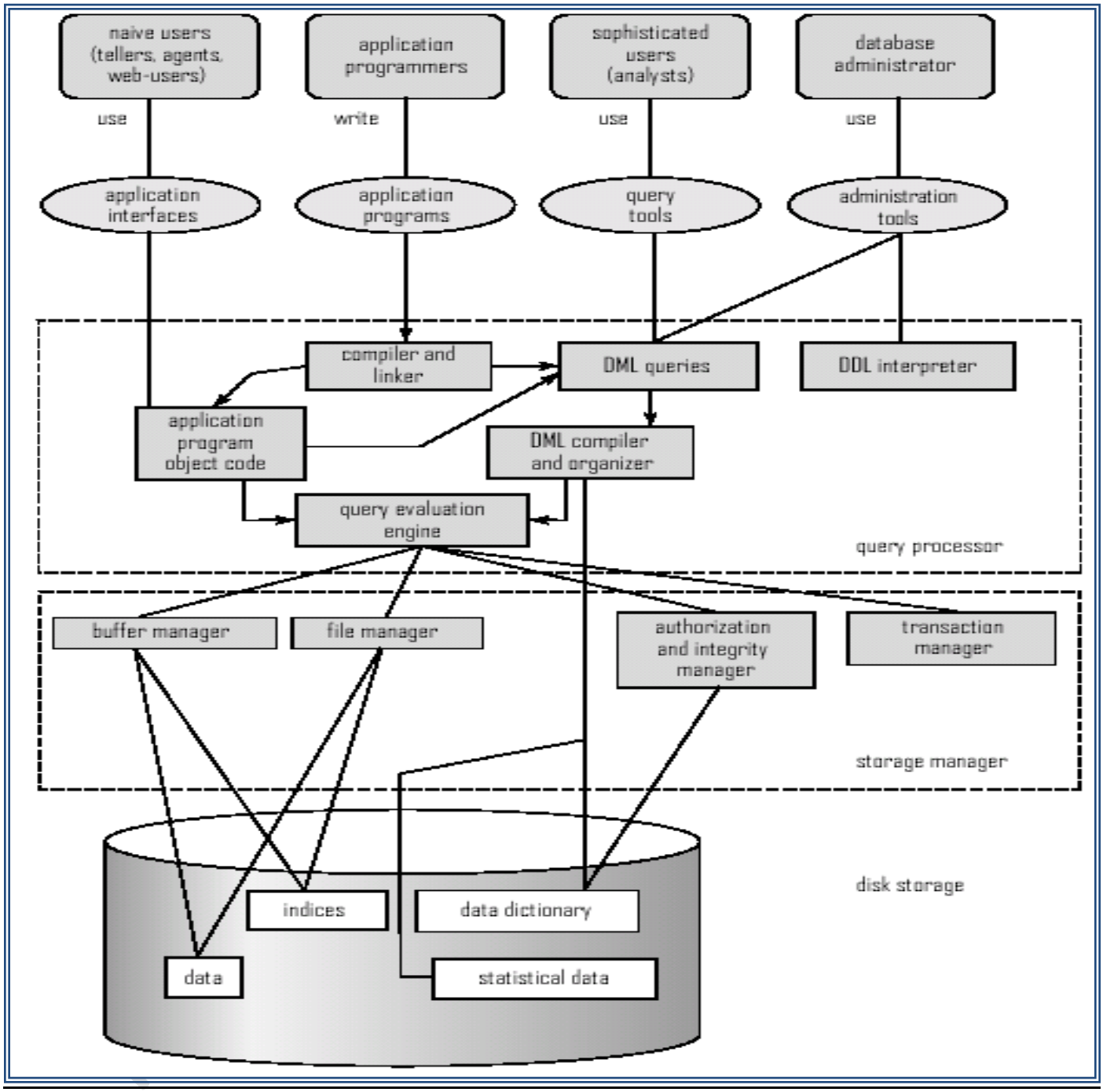
Disadvantages:

1. **Performance**: A major constraint and therefore disadvantage in the use of relational database system is machine performance. If the number of tables between which relationships to be established are large and the tables themselves effect the performance in responding to the sql queries.

2. **Physical Storage Consumption**: With an interactive system, for example an operation like join would depend upon the physical storage also. It is, therefore common in relational databases to tune the databases and in such a case the physical data layout would be chosen so as to give good performance in the most frequently run operations. It therefore would naturally result in the fact that the lays frequently run operations would tend to become even more shared.
3. **Slow extraction of meaning from data**: if the data is naturally organized in a hierarchical manner and stored as such, the hierarchical approach may give quick meaning for that data.

Q.Write a Note on Database Architecture.

- **Database Architecture**



Data Storage and Querying

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

The storage manager is important because databases typically require a large amount of storage space. Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data. A gigabyte is approximately 1000 megabytes (actually 1024) (1 billion bytes), and a terabyte is 1 million megabytes (1 trillion bytes). Since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk

storage and main memory as needed. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that the database system structure the data so as to minimize the need

to move data between disk and main memory.

The query processor is important because it helps the database system to simplify and facilitate access to data. The query processor allows database users to obtain good performance while being able to work at the view level and not be burdened with understanding the physical-level details of the implementation of the system. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level.

1.7.1 Storage Manager

- The *storage manager* is the component of a database system that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for the interaction with the file manager. The raw data are stored on the disk using the file system provided by the operating system. The storage manager translates the various DML statements into low-level file-system commands.

1.7 Data Storage and Querying

- Thus, the storage manager is responsible for storing, retrieving, and updating data in the database.
- The storage manager components include:
  - **Authorization and integrity manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
  - **Transaction manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
  - **File manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
  - **Buffer manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. The buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.
- The storage manager implements several data structures as part of the physical system implementation:
  - **Data files**, which store the database itself.
  - **Data dictionary**, which stores metadata about the structure of the database, in particular the schema of the database.
  - **Indices**, which can provide fast access to data items. Like the index in this textbook, a database index provides pointers to those data items that hold a particular value. For example, we could use an index to find the *instructor* record with a particular *ID*, or all *instructor* records with a particular *name*.
- Hashing is an alternative to indexing that is faster in some but not all cases.

1.7.2 The Query Processor

- The query processor components include:
  - **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
  - **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
- A query can usually be translated into any of a number of alternative evaluation plans that all give the same result. The DML compiler also performs **query optimization**; that is, it picks the lowest cost evaluation plan from among the alternatives.
  - **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.

Database Users:

Users are differentiated by the way they expect to interact with the system

- **Application programmers** – interact with system through DML calls
- **Sophisticated users** – form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Native users** – invoke one of the permanent application programs that have been written previously
  - Examples, people accessing database over the web, bank tellers, clerical staff Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs.
- Database administrator's duties include:
  - Schema definition
  - Storage structure and access method definition
  - Schema and physical organization modification
  - Granting user authority to access the database
  - Specifying integrity constraints
  - Acting as liaison with users
  - Monitoring performance and responding to changes in requirements

---

Transaction Management

- A **transaction** is a collection of operations that performs a single logical function in a database application. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates.

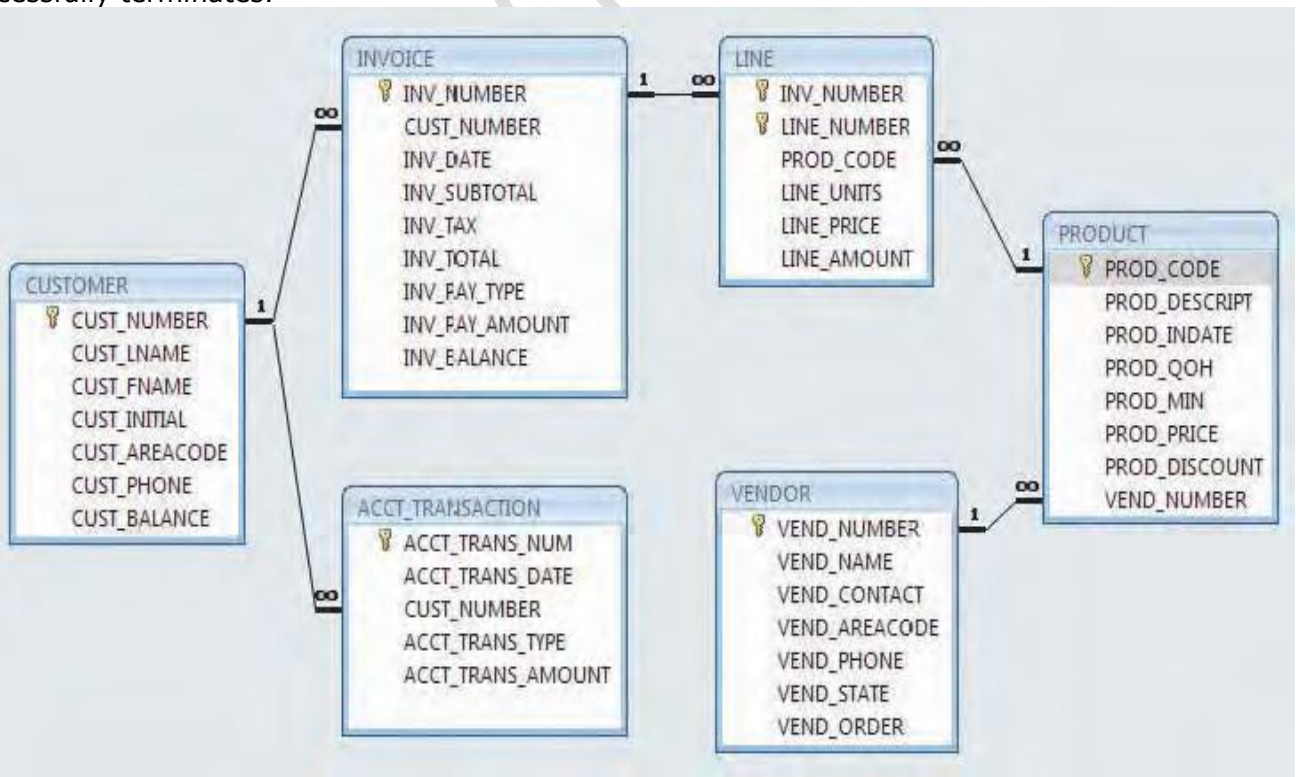


Fig. SaleCo database relational diagram

Transaction Properties

Each individual transaction must display *atomicity*, *consistency*, *isolation*, and *durability*. These properties are sometimes referred to as the ACID test. In addition, when executing multiple transactions, the DBMS must schedule the concurrent execution of the transaction's operations.

The schedule of such transaction's operations must exhibit the property of *serializability*. Let's look briefly at each of the properties.

- **Atomicity** requires that *all* operations (SQL requests) of a transaction be completed; if not, the transaction is aborted. If a transaction T1 has four SQL requests, all four requests must be successfully completed; otherwise, the entire transaction is aborted. In other words, a transaction is treated as a single, indivisible, logical unit of work.
- **Consistency** indicates the permanence of the database's consistent state. A transaction takes a database from one consistent state to another consistent state. When a transaction is completed, the database must be in a consistent state; if any of the transaction parts violates an integrity constraint, the entire transaction is aborted.
- **Isolation** means that the data used during the execution of a transaction cannot be used by a second transaction until the first one is completed. In other words, if a transaction T1 is being executed and is using the data item X, that data item cannot be accessed by any other transaction (T2 ... T*n*) until T1 ends. This property is particularly useful in multiuser database environments because several users can access and update the database at the same time.
- **Durability** ensures that once transaction changes are done (committed), they cannot be undone or lost, even in the event of a system failure.
- **Serializability** ensures that the schedule for the concurrent execution of the transactions yields consistent results. This property is important in multiuser and distributed databases, where multiple transactions are likely to be executed concurrently. Naturally, if only a single transaction is executed, serializability is not an issue.
- A single-user database system automatically ensures serializability and isolation of the database because only one transaction is executed at a time. The atomicity, consistency, and durability of transactions must be guaranteed by the single-user DBMSs. (Even a single-user DBMS must manage recovery from errors created by operating-system-induced interruptions, power interruptions, and improper application execution.)
- Multiuser databases are typically subject to multiple concurrent transactions. Therefore, the multiuser DBMS must implement controls to ensure serializability and isolation of transactions—in addition to atomicity and durability—to guard the database's consistency and integrity. For example, if several concurrent transactions are executed over the same data set and the second transaction updates the database before the first transaction is finished, the isolation property is violated and the database is no longer consistent. The DBMS must manage the transactions by using concurrency control techniques to avoid such undesirable situations.

Transaction Management with SQL

The American National Standards Institute (ANSI) has defined standards that govern SQL database transactions.

Transaction support is provided by two SQL statements: COMMIT and ROLLBACK. The ANSI standards require that when a transaction sequence is initiated by a user or an application program, the sequence must continue through all succeeding SQL statements until one of the following four events occurs:

- A **COMMIT** statement is reached, in which case all changes are permanently recorded within the database. The COMMIT statement automatically ends the SQL transaction.
- A **ROLLBACK** statement is reached, in which case all changes are aborted and the database is rolled back to its previous consistent state.
- The end of a program is successfully reached, in which case all changes are permanently recorded within the database. This action is equivalent to COMMIT.
- The program is abnormally terminated, in which case the changes made in the database are aborted and the database is rolled back to its previous consistent state. This action is equivalent to ROLLBACK.

The use of COMMIT is illustrated in the following simplified sales example, which updates a product's quantity on hand

(PROD_QOH) and the customer's balance when the customer buys two units of product 1558-QW1 priced at $43.99
per unit (for a total of $87.98) and charges the purchase to the customer's account:

```
UPDATE PRODUCT
SET PROD_QOH = PROD_QOH – 2
WHERE PROD_CODE = '1558-QW1';
UPDATE CUSTOMER
SET CUST_BALANCE = CUST_BALANCE + 87.98
WHERE CUST_NUMBER = '10011';
COMMIT;
```

(Note that the example is simplified to make it easy to trace the transaction. In the **Ch10_SaleCo** database, the transaction would involve several additional table updates.)

Actually, the COMMIT statement used in that example is not necessary if the UPDATE statement is the application's last action and the application terminates normally. However, good programming practice dictates that you include the COMMIT statement at the end of a transaction declaration.

A transaction begins implicitly when the first SQL statement is encountered. Not all SQL implementations follow the ANSI standard; some (such as SQL Server) use transaction management statements such as:

```
BEGIN TRANSACTION;
```

to indicate the beginning of a new transaction. Other SQL implementations allow you to assign characteristics for the transactions as parameters to the BEGIN statement. For example, the Oracle RDBMS uses the SET TRANSACTION statement to declare a new transaction start and its properties.

Quick Revision

- A **database-management system** (DBMS) consists of a collection of interrelated data and a collection of programs to access that data. The data describe one particular enterprise.
- The primary goal of a DBMS is to provide an environment that is both convenient and efficient for people to use in retrieving and storing information.
- Database systems are ubiquitous today, and most people interact, either directly or indirectly, with databases many times every day.
- Database systems are designed to store large bodies of information. The management of data involves both the definition of structures for the storage of information and the provision of mechanisms for the manipulation of information.
- In addition, the database system must provide for the safety of the information stored, in the face of system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.
- A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.
- Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and data constraints.
- The relational data model is the most widely deployed model for storing data in databases. Other data models are the object-oriented model, the object relational
- model, and semi structured data models.
- A **data-manipulation language (DML)** is a language that enables users to access or manipulate data. Nonprocedural DMLs, which require a user to specify only what data are needed, without specifying exactly how to get those data, are widely used today.
- A **data-definition language (DDL)** is a language for specifying the database schema and as well as other properties of the data.
- Database design mainly involves the design of the database schema. The entity relationship (E-R) data model is a widely used data model for database design. It provides a convenient graphical representation to view data, relationships, and constraints.
- A database system has several subsystems.
- The **storage manager** subsystem provides the interface between the low level data stored in the database and the application programs and queries submitted to the system.
- The **query processor** subsystem compiles and executes DDL and DML statements.
- **Transaction management** ensures that the database remains in a consistent (correct) state despite system failures. The transaction manager ensures that concurrent transaction executions proceed without conflicting.
- The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.
- Database applications are typically broken up into a front-end part that runs at client machines and a part that runs at the back end. In two-tier architectures, the front end directly communicates with a database running at the back end. In three-tier architectures, the back end part is itself broken up into an application server and a database server.

CHAPTER II: DATA MODELS

Topic Covered:

The importance of data models, Basic building blocks, Business rules, the evolution of data models, Degrees of data abstraction.

A data model is a picture or description which shows how the data is to be arranged to achieve a given task.

- It is a clear model which specifies how the data items are arranged in a given model.
- Some data models which gives a clear picture which shows the manner in which the data records are connected or related within a file structure. These are called structural data models.
- DBMS organize and structure data so that it can be retrieved and manipulated by different users and application programs.
- The data structures and access techniques provided by a particular DBMS are called its data model.
- A data model determined both the personality of a DBMS and the applications for which it is particularly well suited.

Advantages

1. **Simplicity:**

Since the database is based on the hierarchical structure, the relationship between the various layers is logically simple.

2. **Data Security:**

Hierarchical model was the first database model that offered the data security that is provided by the DBMS.

3. **Data Integrity:**

Since it is based on the parent child relationship, there is always a link between the parent segment and the child segment under it.

4. **Efficiency:** It is very efficient because when the database contains a large number of 1:N relationship and when the user require large number of transaction.

Disadvantages

1. **Implementation complexity:**

Although it is simple and easy to design, it is quite complex to implement.

2. **Database Management Problem:** If you make any changes in the database structure, then you need to make changes in the entire application program that access the database.
3. **Lack of Structural Independence**: there is lack of structural independence because when we change the structure then it becomes compulsory to change the application too.
4. **Operational Anomalies**: Hierarchical model suffers from the insert, delete and update anomalies, also retrieval operation is difficult.

Basic Building Blocks:

The basic building blocks of all data models are entities, attributes, relationships, and constraints.

Entity:

- An **entity** is anything (a person, a place, a thing, or an event) about which data are to be collected and stored.
- An entity represents a particular type of object in the real world. Because an entity represents a particular type of object, entities are "distinguishable"—that is, each entity occurrence is unique and distinct.
- For example, a CUSTOMER entity would have many distinguishable customer occurrences, such as ajay, pravin, aniket, etc. Entities may be physical objects, such as customers or products, but entities may also be abstractions, such as flight routes or musical concerts.

Attribute:

- An **attribute** is a characteristic of an entity.
  - For example, a CUSTOMER entity would be described by attributes such as customer last name, customer first name, customer phone, customer address, and customer credit limit.
  - Attributes are the equivalent of fields in file systems.

- An entity is represented by a set of attributes, that is descriptive properties possessed by all members of an entity set.
- **Domain** – the set of permitted values for each attribute Attribute types:
- ***Simple* and *composite* attributes.**
- *Single-valued* and *multi-valued* attributes
- Example: multivalued attribute: *phone_numbers*
- ***Derived* attributes**
- Can be computed from other attributes
- Example: age, given date_of_birth

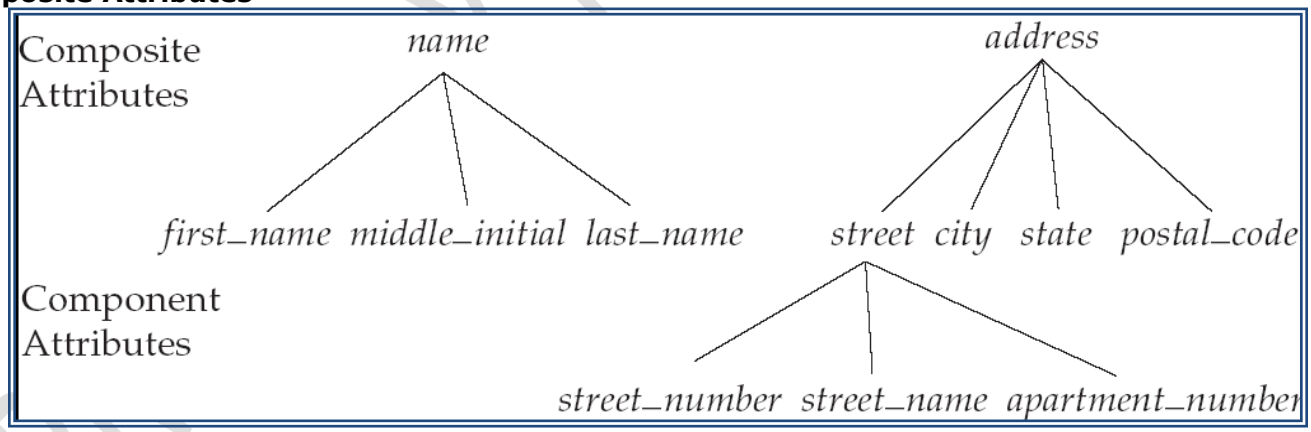
- **Simple and composite attributes**. In our examples thus far, the attributes have been simple; that is, they have not been divided into subparts.
- **Composite** attributes, on the other hand, can be divided into subparts (that is, other attributes). For example, an attribute *name* could be structured as a composite attribute consisting of *first name*, *middle initial*, and *last name*.

Using composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions, and to only a component of the attribute on other occasions.

Note also that a composite attribute may appear as a hierarchy. In the composite attribute *address*, its component attribute *street* can be further divided into *street number*, *street name*, and *apartment number*. Figure 7.4 depicts these examples of composite attributes for the *instructor* entity set.

- **Single-valued and multivalued attributes.**
  - The attributes in our examples all have a single value for a particular entity. For instance, the *student ID* attribute for a specific student entity refers to only one student *ID*. Such attributes are said to be **single valued**. There may be instances where an attribute has a set of values for a specific entity. Suppose we add to the *instructor* entity set

Composite Attributes



a *phone number* attribute. An *instructor* may have zero, one, or several phone numbers, and different instructors may have different numbers of phones.

This type of attribute is said to be **multivalued**. As another example, we could add to the *instructor* entity set an attribute *dependent name* listing all the dependents. This attribute would be multivalued, since any particular instructor may have zero, one, or more dependents.

To denote that an attribute is multivalued, we enclose it in braces, for example {*phone number*} or {*dependent name*}.

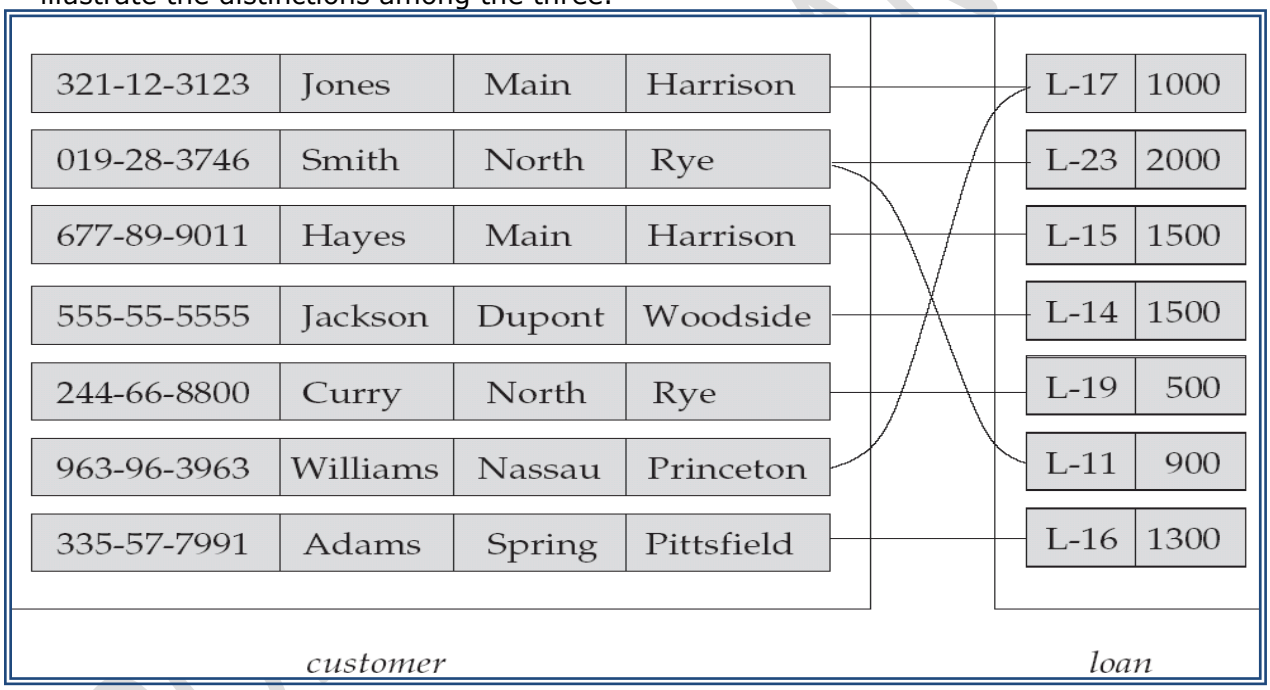
Where appropriate, upper and lower bounds may be placed on the number of values in a multivalued attribute. For example, a university may limit the number of phone numbers recorded for a single instructor to two. Placing bounds in this case expresses that the *phone number* attribute of the *instructor* entity set may have between zero and two values.

- **Derived attribute.**

- The value for this type of attribute can be derived from the values of other related attributes or entities. For instance, let us say that the *instructor* entity set has an attribute *students advised*, which represents how many students an instructor advises. We can derive the value for this attribute by counting the number of *student* entities associated with that instructor.
- As another example, suppose that the *instructor* entity set has an attribute *age* that indicates the instructor's age. If the *instructor* entity set also has an attribute *date of birth*, we can calculate *age* from *date of birth* and the current date. Thus, *age* is a derived attribute. In this case, *date of birth* may be referred to as a *base* attribute, or a *stored* attribute. The value of a derived attribute is not stored but is computed when required.

Relationship:

- A **relationship** describes an association among entities.
- For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use the shorthand notations 1:M or 1..*, M:N or *..*, and 1:1 or 1..1, respectively. (Although the M:N notation is a standard label for the many-to-many relationship, the label M:M may also be used.) The following examples illustrate the distinctions among the three.



Constraints:

A **constraint** is a restriction placed on the data. Constraints are important because they help to ensure data integrity.

Constraints are normally expressed in the form of rules. For example:

_ An employee's salary must have values that are between 6,000 and 350,000.

_ A student's GPA must be between 0.00 and 4.00.

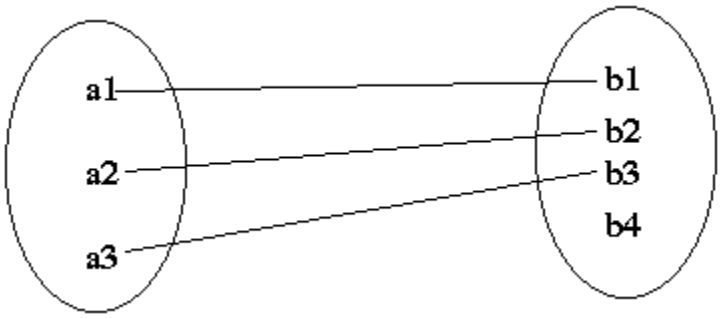
_ Each class must have one and only one teacher.

How do you properly identify entities, attributes, relationships, and constraints? The first step is to clearly identify the business rules for the problem domain you are modeling.

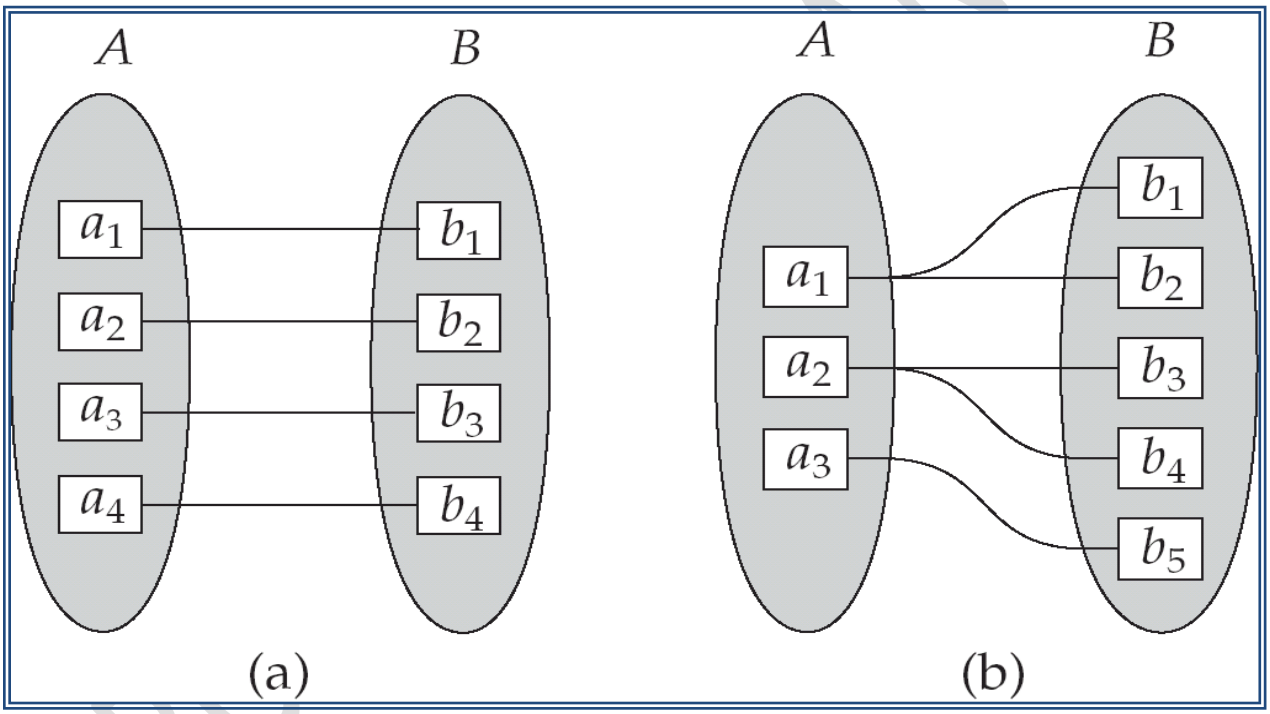
- **Mapping Cardinalities**

The relationship set *advisor*, between the *instructor* and *student* entity sets may be One-to-one, one-to-many, many-to-one, or many-to-many. To distinguish among these types, we draw either a directed line (→) or an undirected line (—) between the relationship set and the entity set in question, as follows:

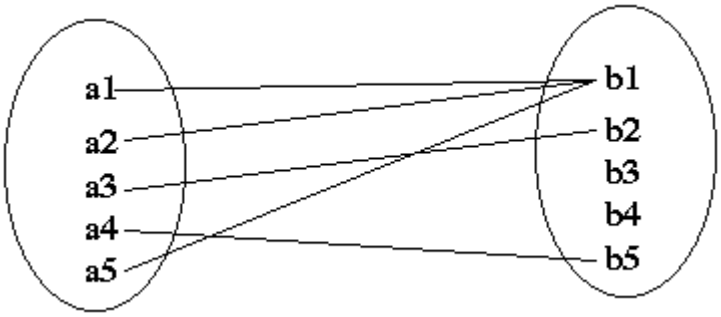
• **One-to-one:** We draw a directed line from the relationship set *advisor* to both entity sets *instructor* and *student* (see Figure 7.9a). This indicates that an instructor may advise at most one student, and a student may have at most one advisor.



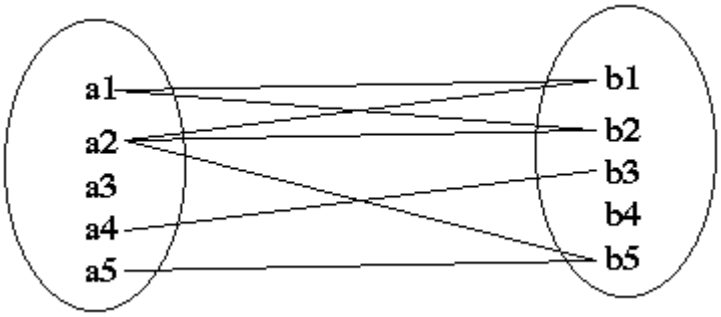
- **One-to-many:** We draw a directed line from the relationship set *advisor* to the entity set *instructor* and an undirected line to the entity set *student* (see Figure 7.9b). This indicates that an instructor may advise many students, but a student may have at most one advisor.

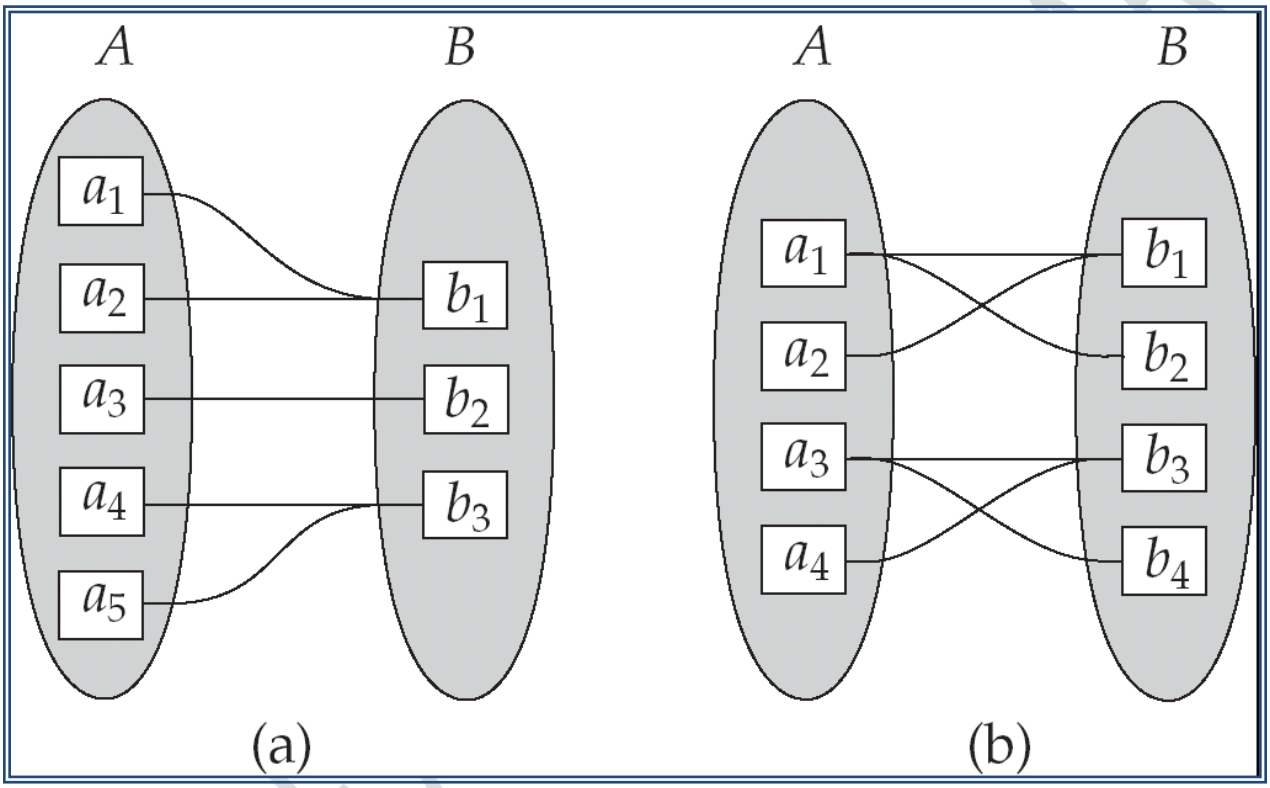


- **Many-to-one:** We draw an undirected line from the relationship set *advisor* to the entity set *instructor* and a directed line to the entity set *student*. This indicates that an instructor may advise at most one student, but a student may have many advisors.



• **Many-to-many:** We draw an undirected line from the relationship set *advisor* to both entity sets *instructor* and *student* (see Figure 7.9c).





This indicates that an instructor may advise many students, and a student may have many advisors.

Participation Constraints

- The participation of an entity set *E* in a relationship set *R* is said to be **total** if every entity in *E* participates in at least one relationship in *R*. If only some entities in *E* participate in relationships in *R*, the participation of entity set *E* in relationship *R* is said to be **partial**. In Figure 7.5a, the participation of *B* in the relationship set is total while the participation of *A*in the relationship set is partial. In Figure 7.5b, the participation of both *A*and *B* in the relationship set are total.
- For example, we expect every *student* entity to be related to at least one instructor through the *advisor* relationship. Therefore the participation of *student* in the relationship set *advisor* is total. In contrast, an *instructor* need not advise any students. Hence, it is possible that only some of the *instructor* entities are related to the *student* entity set through the *advisor* relationship, and the participation of *instructor* in the *advisor* relationship set is therefore partial.

Business Rules:

- A **business rule** is a brief, precise, and unambiguous description of a policy, procedure, or principle within a specific organization. In a sense, business rules are misnamed: they apply to *any* organization, large or small—a business, a government unit, a religious group, or a research laboratory—that stores and uses data to generate information.
- Business rules, derived from a detailed description of an organization's operations, help to create and enforce actions within that organization's environment. Business rules must be rendered in writing and updated to reflect any change in the organization's operational environment.
- Properly written business rules are used to define entities, attributes, relationships, and constraints. Any time you see relationship statements such as "an agent can serve many customers, and each customer can be served by only one agent," you are seeing business rules at work. You will see the application of business rules throughout this book, especially in the chapters devoted to data modeling and database design.
- Business rules are the rules that are created to affect the way your business works. Usually, these are rules that involve employees or staff and are rules that specify what they can and cannot do.
- A great example of a business rule involves marriages. For many companies, a boss is not allowed to marry an employee or an accountant at a company is usually not allowed to marry another accountant.
- In this case, the accountants are not allowed to be married because there is a more
- likely chance that the spouses can change financial information and then cover for one another.
- These rules are intended to prevent disruption in a company or business.
- Business Rules are used every day to define entities, attributes, relationships and
- constraints.
- Usually though they are used for the organization that stores or uses data to be an explanation of a policy, procedure, or principle.
- The data can be considered significant only after business rules are defined, without them it's just records, but to a business they are the characteristics that are defined and seen by the company.
- Business Rules help employees focus on and implement the actions within the organizations environment.
- Some things to think about when creating business rules are to keep them simple, easy to understand, keep them broad so that everyone can have a similar interpretation. To be considered true, business rules must be in writing and kept up to date.
- Identifying business rules are very important to the database design.
- Business rules allow the creator to develop relationship participation rules and constraints and to create a correct data model.
- They also allow the creators to understand business processes, and the nature, role and scope of the data.
- They are a communication tool between users and creators, and they also help standardize the company's view of the data.
- It is important to keep in mind that some business rules cannot be modeled.
- Business Rules give the proper classification of entities, attributes, relationships, and constraints.
- Sources of business rules are managers, policy makers, department managers, written documentation, procedures, standards, operation manuals, and interviews with end users.
- Some examples of business
- rules:
- Departments ------offers---------Course
- Course----------generates---------Class
- Professor--------teaches----------Class
- There are several protocols to the way business rules are written. Not every protocol has to be followed, but in general, a well written set of business rules consist of having a unique identifier, describes one and only one concept, are written in plain language, are written, and are from a single source.
- In terms of a unique identifier, business rules should come with an identifier that may consist of the rule number and the department it affects. And example would be 'BRacc01'. In this case, this business rule (BR) is directly related to the accounting department.

- Another important aspect of business rules consist of how the rules are shared within the company.
- A protocol for business rules that many follow is that the business rules are written down. However, with many businesses sharing information directly over the internet, some
- are opting to place their business rules online in company blogs, wikis, and websites.
- This shares the business rules with all employees faster and easier. In relation to how business rules are shred, it is very important that business rules are written in plain language.
- If business rules are written at a high level language, there is an increased chance that not every person will understand what the business rules cover or what is acceptable and what is not.

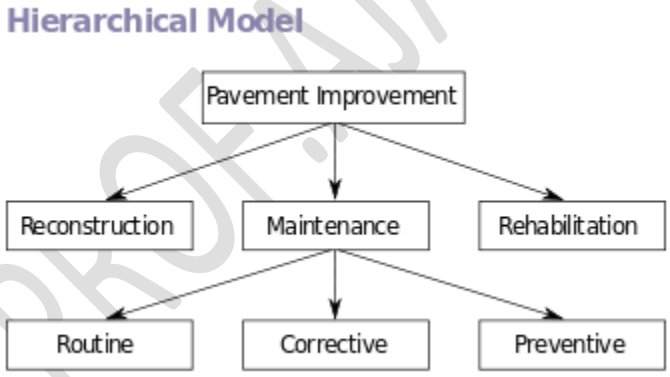
Types of Data Models:

There are four different types of data models
Hierarchical databases
Network databases
Relational databases
Object oriented databases

Hierarchical databases

Hierarchical Databases is most commonly used with mainframe systems.

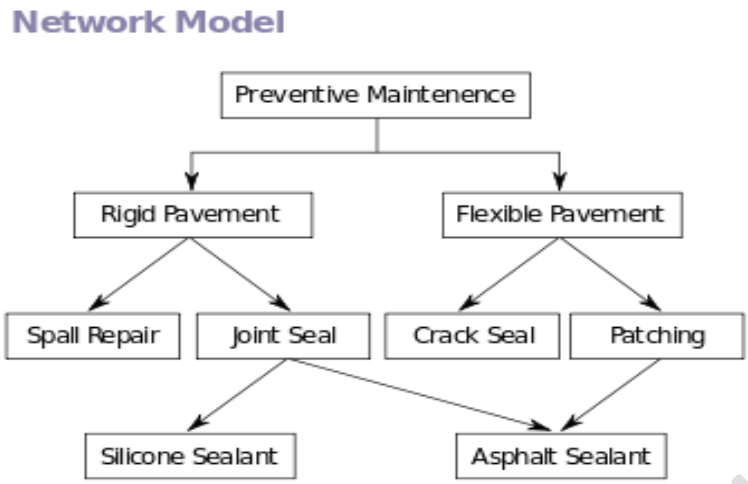
- It is one of the oldest methods of organizing and storing data and it is still used by some organizations for making travel reservations.
- A hierarchical database is organized in pyramid fashion, like the branches of a tree extending downwards.
- In this model, related fields or records are grouped together so that there are higher
- -level records and lower
- -level records, just like the parents in a family tree sit above the subordinated children.
- Based on this analogy, the parent record at the top of the pyramid is called the root record .
- A child record always has only one parent record to which it is linked, just like in a normal family tree.
- In contrast, a parent record may have more than one child record linked to it. Hierarchical databases work by moving from the top down.
- A record search is conducted by starting at the top of the pyramid and working down through the tree from parent to child until the appropriate child record is found. Furthermore, each child can also be a parent with children underneath it.



Network databases

- Network databases are similar to hierarchical databases by also having a hierarchical structure. There are a few key differences, however.
- Instead of looking like an upside-down tree, a network database looks more like a cobweb or interconnected network of records.
- In network databases, children are called members and parents are called owners.
- The most important difference is that each child or member can have more than one parent (or owner).
- Similar to hierarchical databases, network databases are principally used on mainframe computers.

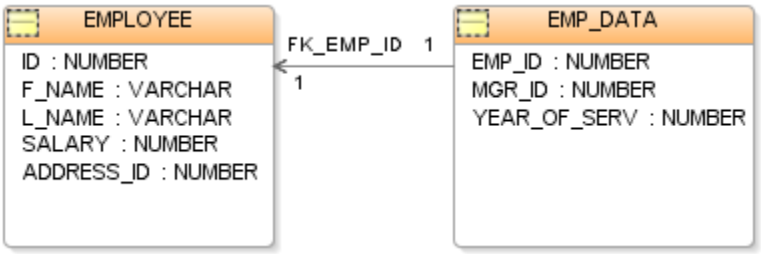
- Since more connections can be made between different types of data, network databases are considered more flexible.
- However, two limitations must be considered when using this kind of database.
- Similar to hierarchical databases, network databases must be defined in advance. There is also a limit to the number of connections that can be made between records.



Relational databases

- Pre-relational models depended upon being able to determine explicitly where and how individual records were stored.
- Early relational proponents argued that the relational data model viewed information
- logically rather than physically, but this is not quite correct.
- Earlier data models associated the logical and physical aspects of information together; logically-related information was stored in physical proximity within a data file. The relational data model first separated the logical from the physical aspects.
- The relational data model looks at information as an unordered collection of "relations."
- Each relation is populated with unordered "tuples" of the same unordered "field" structure.
- Fields may only contain values of a well-defined ("atomic") domain or the null value. The unordered aspect needs to be emphasize
- d. For expository purposes, relations are often viewed as "tables".
- The tuples constitute the "rows" of the table; values for a specific field constitute "columns". However, the "table data model" tends to impose a very non- relational ordering on both tuples and fields. Relations are an abstraction of how data is stored; tables are just one of many possible implementations.
- Some of the relational terms are crafted to emphasize the distinction between logical and physical features, to avoid confusing one concept with another. However, vocabulary leakage from other disciplines has sprinkled into the conversation of relational proponents.
- There is a strong tendency to refer to an individual tuple/row as a "record" because collections of fields in other models are called records. "Attribute" is often used synonymously with field.
- To be sure, "unordered" implies neither "chaotic" nor "random". Relations and Fields are named uniquely and identified easily.
- Distinguishing between tuples is more subtle since the order is not pre-defined.
- Rather than depending upon relative (as in hierarchy) or absolute (as in network) locations, tuples may only be differentiated according to their contents.
- Consequently, duplicate tuples are not permitted within a single relation.
- Even more strongly, distinct tuples must have a unique "key" (some combination of a relation's named fields).

- The set of minimal keys includes one "primary key"; the rest are "candidate keys". Within a tuple, references to other tuples are expressed as a "foreign key," which should contain the values of the referenced tuple's primary key.
- Relational theory provides a firm mathematical foundation for data management. Set theory could be applied to relations using relational algebraic operations(union,intersection, join,projection, etc.).
- Assertions about the existence or non-existence of some condition with a data base could be proven with a rigor unachievable with earlier models.



Object oriented databases

- A data model is a logic organization of the real world objects(entities), constraints on them, and the relationships among objects.
- A DB language is a concrete syntax for a data model.
- A DB system implements a data model.
- A core object-oriented data model consists of the following basic object-oriented concepts:
- (1)object and object identifier: Any real world entity is uniformly modeled as an object (associated with a unique id: used to pinpoint an object to retrieve).
- (2)attributes and methods: Here every object has a state (the set of values for the attributes of the object) and a behavior (the set of methods- program code- which operate on the state of the object). The state and behavior encapsulated in an object are accessed or invoked from outside the object only through explicit message passing.
- An attribute is an instance variable, whose domain may be any class: user-defined or primitive. A class composition hierarchy (aggregation relationship) is orthogonal to the concept of a class hierarchy. The link in a class composition hierarchy may form cycles.
- (3)class: a means of grouping all the objects which share the same set of attributes and methods. An object must belong to only one class as an instance of that class (instance –of relationship). A class is similar to an abstract data type. A class may also be primitive (no attributes), e.g., integer, string, Boolean.
- (4)Class hierarchy and inheritance: derive a new class (subclass) from an existing class (superclass). The subclass inherits all the attributes and methods of the existing class
- And may have additional attributes and methods. Single inheritance (class hierarchy) vs. multiple inheritance (class lattice).

Degrees of data abstraction :Refer chapter I

Quick Revision

- A data model is a picture or description which shows how the data is to be arranged to achieve a given task.
- The data structures and access techniques provided by a particular DBMS are called its data model.
- In 1964 the first commercial database management system (DBMS) was developed widely known as Integrated Data Store (IDS).
- A hierarchical database is organized in pyramid fashion, like the branches of a tree extending downwards.
- In hierarchical model, the parent record at the top of the pyramid is called the root record and the leaf node is called the child record.
- Network databases are similar to hierarchical databases by also having a hierarchical structure.
- The relational model organizes the records and stores the records in rows and columns.

CHAPTER III: DATABASE DESIGN AND ER-DIAGRAM

Topic Covered: Database Design, ER-Diagram and Unified Modeling Language

---

The database design process consists of a number of steps listed below. We will focus mainly on step 2, the conceptual database design, and the models used during this step.

Step 1:
Requirements Collection and Analysis

- Prospective users are interviewed to understand and document data requirements
- This step results in a concise set of user requirements, which should be detailed and complete.
- The functional requirements should be specified, as well as the data requirements. Functional requirements consist of user operations that will be applied to the database, including retrievals and updates.
- Functional requirements can be documented using diagrams such as sequence diagrams, data flow diagrams, scenarios, etc.

Step 2:
Conceptual Design

- Once the requirements are collected and analyzed, the designers go about creating the conceptual schema.
- Conceptual schema: concise description of data requirements of the users, and includes a detailed description of the entity types, relationships and constraints.
- The concepts do not include implementation details; therefore the end users easily understand them, and they can be used as a communication tool.
- The conceptual schema is used to ensure all user requirements are met, and they do not conflict.

Step 3: Database Implementation

- Many DBMS systems use an implementation data model, so the conceptual schema is transformed from the high-level data model into the implementation data model.
- This step is called logical design or data model mapping, which results in the implementation data model of the DBMS.

Step 4: Physical Design

- Internal storage structures, indexes, access paths and file organizations are specified.
- Application programs are designed and implemented ER Model
- In software Engineering, an entity relational model is an abstract and conceptual representation of data Entity-relationship modeling is a database modeling method, used to produce a type of conceptual schema or semantic data model of a system, often
- A relational database and its requirements in a top-down fashion.
- Diagrams created by this process are called entity-relationship diagrams,

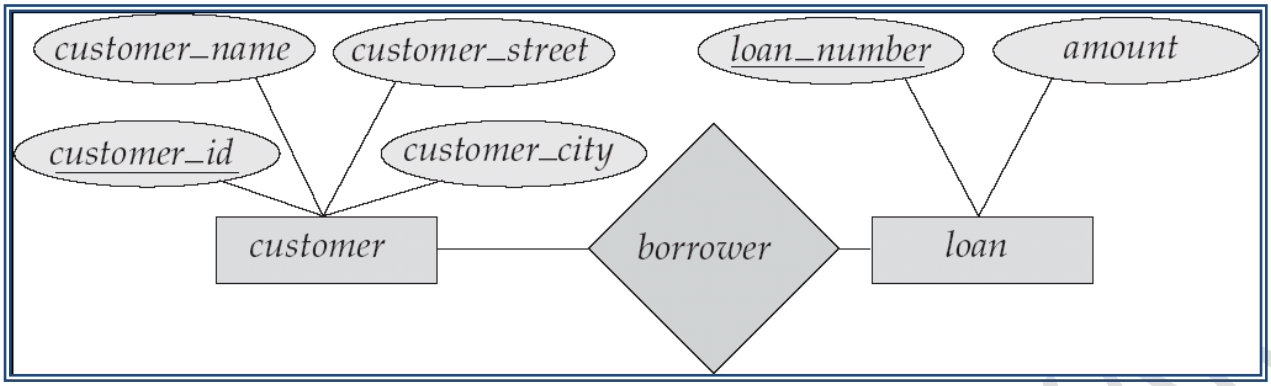
ER diagrams or ERDs.

In 1976, Entity relationship model developed by Chen, ER Model is high level Conceptual model which used Conceptual design of database where as relational model are used to logical design of database

ER Diagram

•A database can be modeled as A collection of entities Relationship among the entities
•An entity is an real world object that exist and it is distinguishable from other entities
Example Person, company, event, plant
•All the entities in the data model have attributes as known as properties of an entities
Example: people have names and addresses
An Entity set is a set of entities of all same type that share the same properties.
Example: set of all persons, companies, trees, holidays

ER Diagram



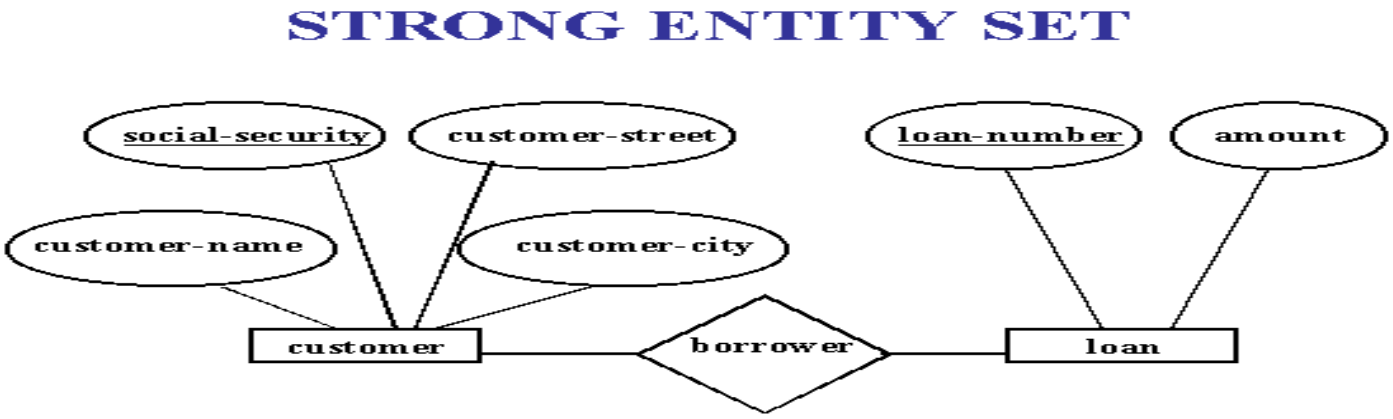
- Rectangles represent entity sets.
- Diamonds represent relationship sets.
- Lines link attributes to entity sets and entity sets to relationship sets.
- Underline indicates primary key attributes
- Ellipses represent an attributes
- Double Lines represent total participation of an entity in a relationship set
- Double rectangle represent a weak entity sets

Strong Entity type

- An entity type which has own distinct primary key that used to identify specific uniquely from another entity type is called as Strong Entity type
- An Entity type which is independent on some other entity type is called Strong Entity type

Example

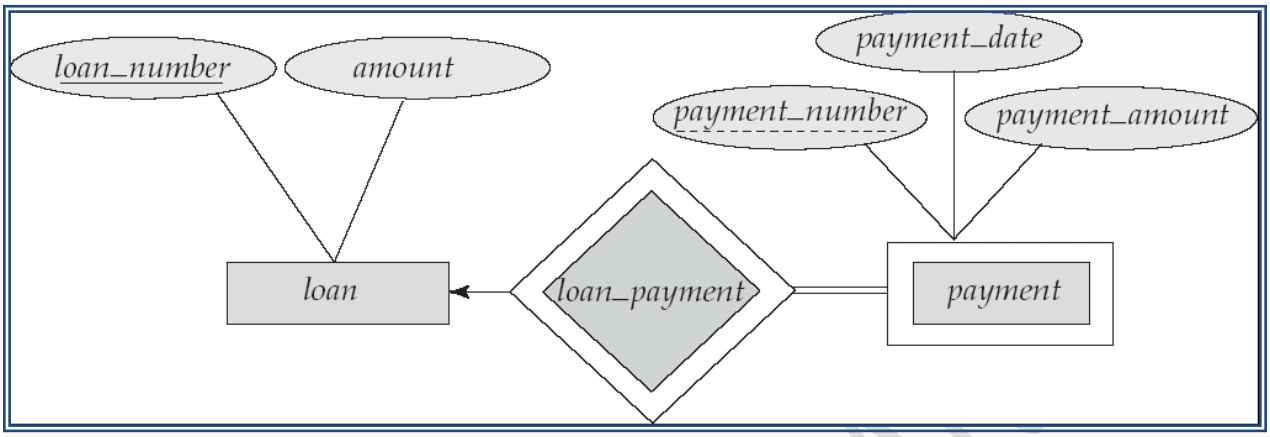
- In the Case of Client entity Client_no is the primary key of Client entity which is used to uniquely identified among the Client 's entity set
- In the case of Customer Entity , Customer_id is the primary key of Customer Entity which is used to uniquely identified among the Customer's entity set
- Strong Entity type is represented by rectangle Symbol



Weak entity Type

- Entity type which is dependent on some other entity type is called as Weak entity type.
- Weak entity type is dependent on a strong entity and cannot exist on its own.
- It does not have a unique identifier that has partial identifier.
- Partial identifier is represented by double-line.

- Some weak entities assign partial identifiers and such partial identifiers of an weak entity called as discriminator.
- Weak entity type is represented by double rectangle.
- Identify relationship
- Strong entity type is link with the weak entity type



Difference between Weak and Strong entity type

Strong Entity Set	Weak Entity Set
it has its own primary key.	It does not save sufficient attributes to form a primary Key on its own.
It is represented by a rectangle.	It is represented by a double rectangle.
It contains a primary key represented by an underline.	It contains a Partial Key or discriminator represented by a dashed underline.
The member of strong entity set is called as dominant entity set.	The member of weak entity set is called as subordinate entity set.
The Primary Key is one of its attributes which uniquely Identifies its member.	The Primary Key of weak entity set is a combination of partial Key and Primary Key of the strong entity set.
The relationship between two strong entity set is represent by a diamond symbol.	The relationship between one strong and a weak entity set is represented by a double diamond sign. It is known as identifying relationship.
The line connecting strong entity set with the relationship is single	The line connecting weak entity set with the identifying relationship is double.
Total participation in the relationship may or may not exist.	Total participation in the identifying relationship always exists.

Attributes

Properties of an entity or relationship type is called as attribute

Example

Staffno, staffname,staff_designation is describes an entity Staff Value of an attribute play a major role of data stored in database

Each entity will have the value which is assigned to its attributes

Consider an example

Above stated example of Staff Entity which has the attribute named as staffno, the value which is assigned to the staff attribute is '101' and the staffname attribute has the value is 'Mahendra, and staff_designation attribute has the value is 'Manager'

Attribute domains

The set of allowable values which is assigned to one or more attribute is known as Attribute domains

There are types of attributes has been classified Such as simple and Composite type, single valued and multi valued attributes Stored and derived attributes, null attributes and Key Attributes.

1) Simple Attributes

Simple attributes is an attributes which can further divided into two parts

OR

An Attribute composed of single components with an independent existence

For an example: Designation of an staff and Salary of an staff

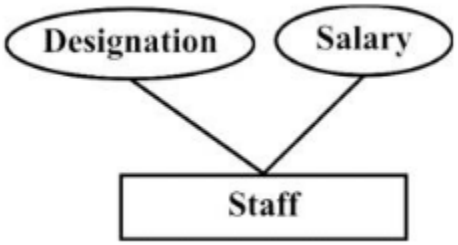


Fig. Simple Attributes

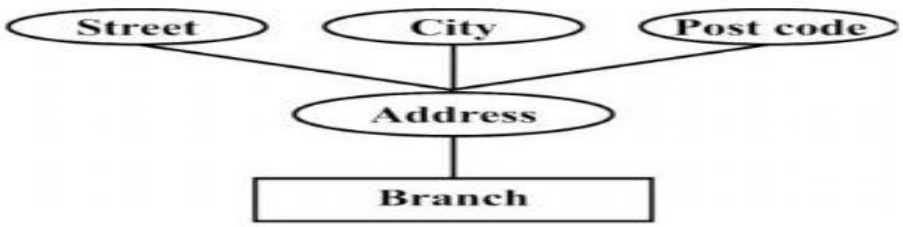
Composite Attribute

Composite Attribute is an attribute which is further divided into many parts.

An attributed composed of multiple components each component has its own independent existence.

Example

Address attributes of an Branch entity that can be further divided in to sub parts i.e. street, city and postal code as an attributes.



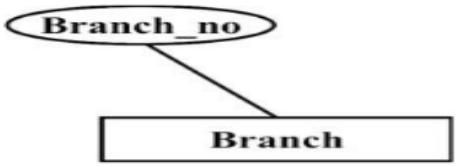
Composite Attributes

2) Single valued and Multi Valued attributes

Single valued attribute is an attribute which as single value(atomic) for each entity.

An attribute that holds a single value for each occurrence of an entity type

Example: Each branch has only single valued attributes is known as branch_no



Single Valued attributes

Multivalued attributes

Multivalued attribute is an attributes which as many values for each entity
An attribute that holds multiple values for each occurrence of an entity type.
Example: Each staff member has multiple mobile numbers

Multivalued Attributes

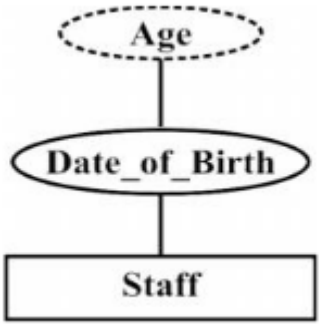
3) Stored and Derived attributes

Stored attributes is an attribute which is used supplied a value to the related attribute
Example Date_of_birth of an staff is a stored attributes

Derived attributes

The value from the derived attribute is derived from the stored attribute for an example Date_of_birth is a stored attribute for an each staff member. The value for an Age can be derived from the Date_of_birth attributes i.e. by subtracting the Date_of_birth from the Current date, therefore the Stored attributes is used

Supplied a value to the related attributes



Null attribute

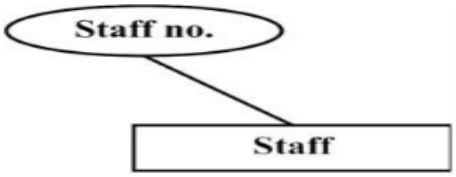
The attribute which take NULL value when entity does not have the value to it.
The Null attribute is an attribute their value is unknown, unassigned and missing information

Key Attributes

This attribute has the unique value for an entity which is used to identified given row in the table is called as key attribute of an entity

Example:

Staff_ no is an key attribute which has an unique value which is used to identifies given row in the table

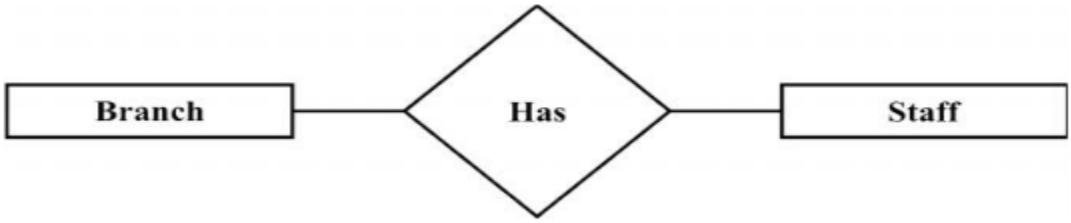


Relationships

A set of meaningful relationship among several entities

We used to indicate the diamond symbol for Relationships among the several entities, it could read from left to right

Example: Branch has a staff



Degree of relationship

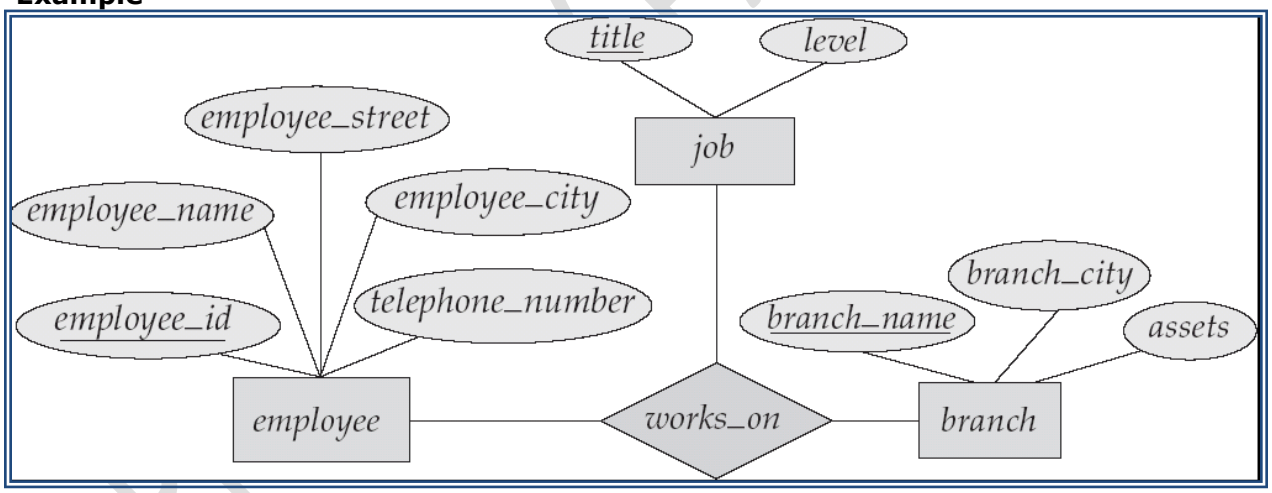
It is the number of entities participated in a particular relational model

There are two type of degree of relationship.

Binary relationship: A Relationship of degree two is called as binary relationship

Ternary Relationship: A relationship of degree three is called as Ternary relationship.

Example



E-R Diagram with a Ternary Relationship

Relationship set

The collection of similar relationship is known as Relationship set

Constraints on relationship

1)Mapping Constraints / Cardinalities

The number (or range) of possible entity type that is associated to another entity type through a particular entity

Cardinalities indicates that a specific number of entity occurrence of related entity .

Type of Mapping Constraints

One-to-one (1:1)

One-to-many (1:*)

Many-to-one(*:1)
Many-to-many (*:*)

One-to-one (1:1)

In this type of Mapping Constraint One record of an entity is related to the one record of an another entity

That is one row on an table is related to an one row of another table i.e.

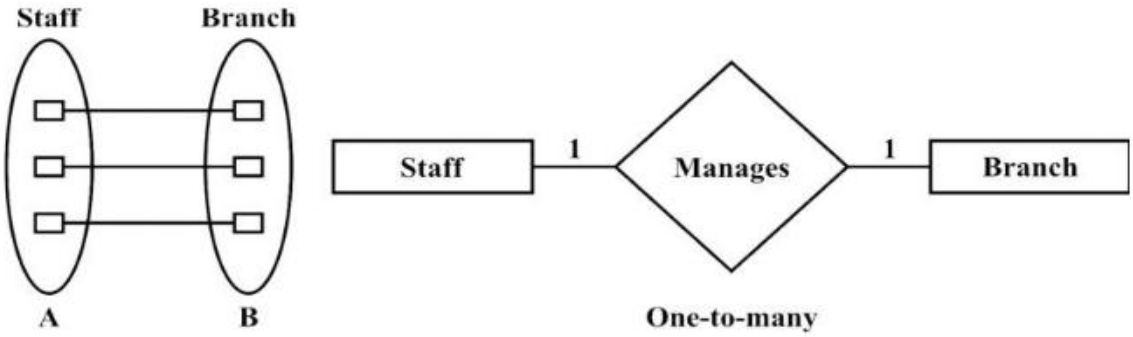
A is associated with at most one entity in B and B is associated with at most one entity in A

Example

Each branch is managed by one member of the staff that's means

Branch Manager

A member of staff can manage zero or one branch



2)**One-to-many(1:*)**

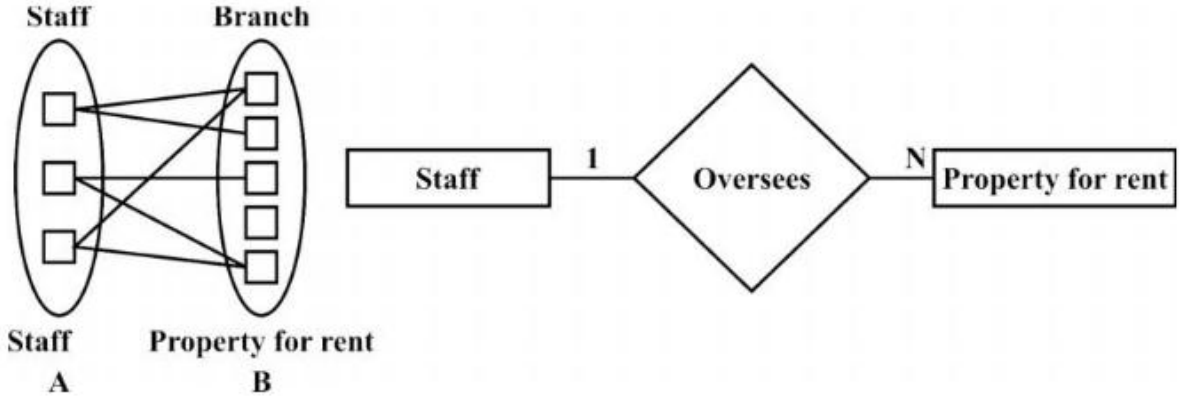
In this constraints, One record in the entity can be related with many record in other entity

A is associated with any number of entities in B

B is associated with at most one entity in A

E.g. each member of staff oversees zero or more prosperity for rent

Every row in the Staff table can have relationship with many rows in the properityforRent Table

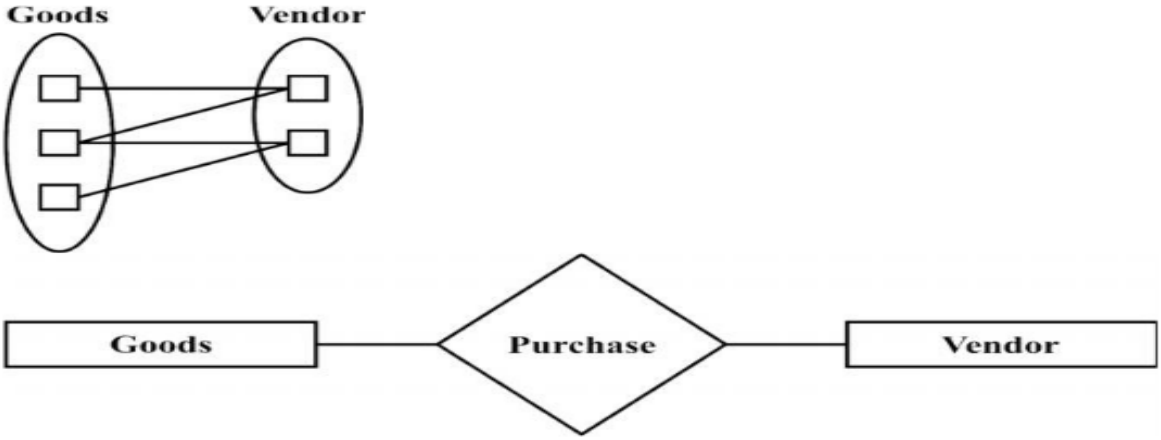


Many To One

In this type Mapping Constraints, Many records in the one entity is related to the only one records in the other entity

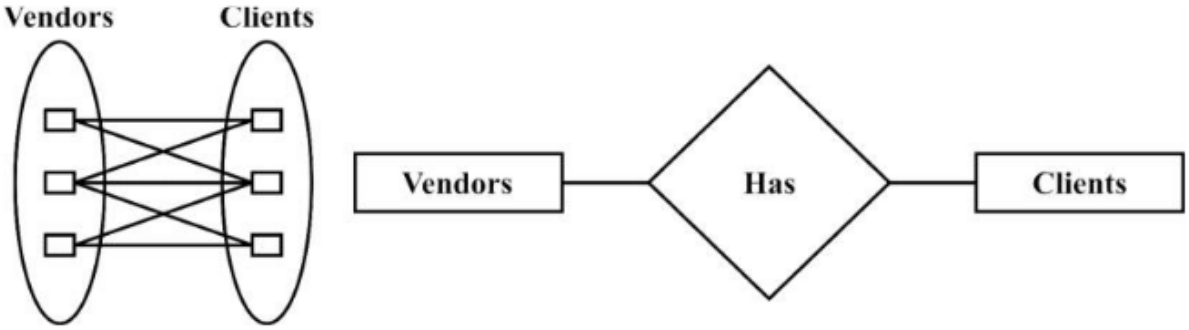
An entity in A is associated with at least one entity in B. an entity in B can be associated with any number of entities in A.

Example one vendors has many Goods and Many Goods is purchase by one Vendors



Many to Many

In this Mapping Constraints, Many records in the entity is related
Many records in the other entity
An entity in A is associated with any number of entities in B. and an entity in B is associated with any number of entities in A.
Many Vendors Has Clients and Many Clients has may Vendors



Participation Constraints

There are two types of participation constraints:
Total Participation: Every Instance of the first Entity type must share with one or more instances of the relationship type with the other entity type.
The total participation is represented by a dark line or double line between the relationship and entity

Every Branch office is allocated members of Staff
Partial Participation: There exist an instance of the first entity type that don't share an instance of the relationship type with the other entity type.

A member of Staff need not work at a Branch office
Notations used In ER Diagrams For Representing Relations
1) Cardinality Ratio Notation
In this method, Cardinality ratio (of a binary relationship): 1:1, 1:N, N:1, or M:N
Shown by placing appropriate numbers on the relationship edges
13

Every Branch office is allocated members of Staff

Partial Participation

: There exist an instance of the first entity type that don't share an instance of the relationship type with the other entity type.

A member of Staff need not work at a Branch office

Notations used In ER Diagrams For Representing Relations

1) Cardinality Ratio Notation

In this method ,Cardinality ratio (of a binary relationship): 1:1, 1:N,N:1, or M:N

Shown by placing appropriate numbers on the relationship edges.

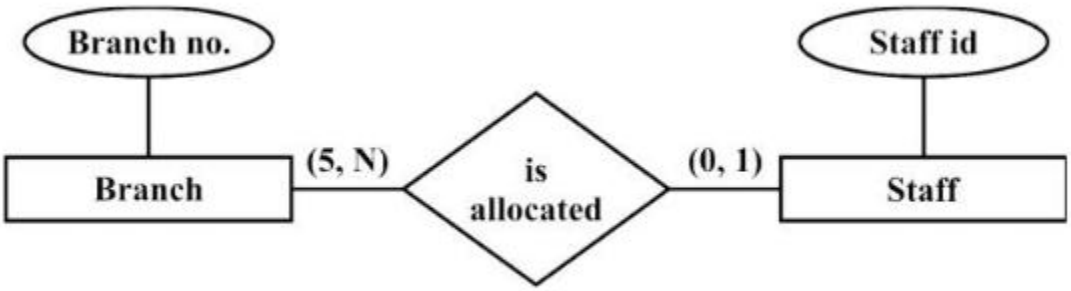
Eg

Number of Staffs working in Branch

2)Min–Max notation

The alternate of notation by specify the pair of integer, that used to specify the minimum and maximum participation of each entity type in the form of(min, max)

The Minimum participation of 0 indicate partial participation whereas maximum participation of 1 or more indicates total participation



At least 5 staff is allocated to branch

Limitation of Entity Relationship Model

Problems may arise when designing a conceptual data model called connection traps.

•Often due to a misinterpretation of the meaning of certain relationships.

Extended Entity Relation Relationship Model

Since 1980 there has been increase in the emergence of new database application with more demanding application Basic concepts of ER modeling are not sufficient to represent the requirement of newer, more complex operation

To overcome the issue of ER modeling there is response in development of additional 'semantic' modeling concept

Semantic concept which is integrated into original ER Model is known as Extended Entity Relation Relationship Model (EER)

Additional Concept which is includes in the Extended Entity Relation Relationship Model are specialization/ generalization, categorization, superclass/subclass, attribute inheritance

Extended EER Model is used the concept of object oriented such as inheritance

Sub classes and Super classes

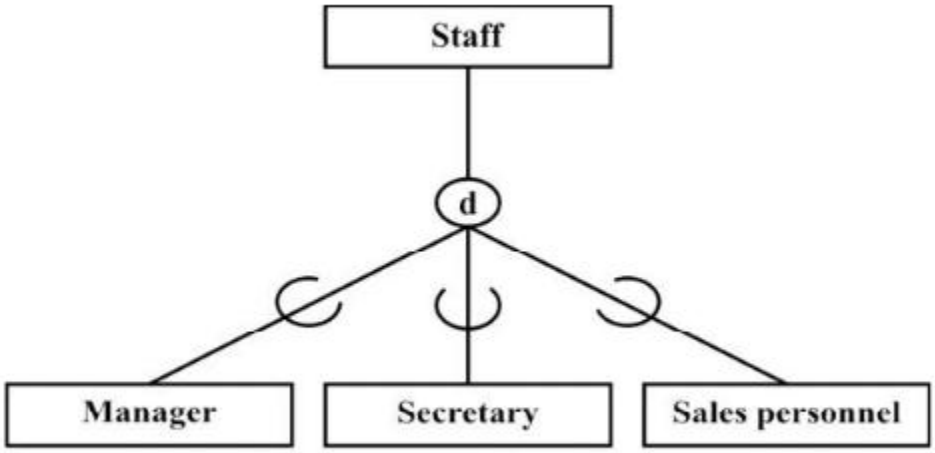
In some case, entity type has numerous sub-grouping of its entities because that are meaningful way for representation and need to be explicitly defined because of their importance

The set listed is a subset of the entities that belong to the staff entity which means that every entity that belongs to one of the sub sets is also an Staff

An entity type that includes distinct Subclasses that require to be represented in a data model is called as super class.

A Subclass is an entity type that has a distinct role and is also a member of the Superclass.

Staff is the super class where as manager, Secretary, sales personnel is the subclass



Superclass / Subclass Relationship

Superclass /Subclass Relationship

The relationship between super class and subclass is called Superclass /Subclass Relationship
In Superclass /Subclass Relationship, the encircled 'd'
Indicates that there is Superclass /Subclass Relationship, it is denoted by the symbol
Hence Superclass /Subclass Relationship lead to the object oriented Concept is called as Inheritance
As the above diagram, Arc drawn above the line towards
Subclass indicated inheritance Relationship

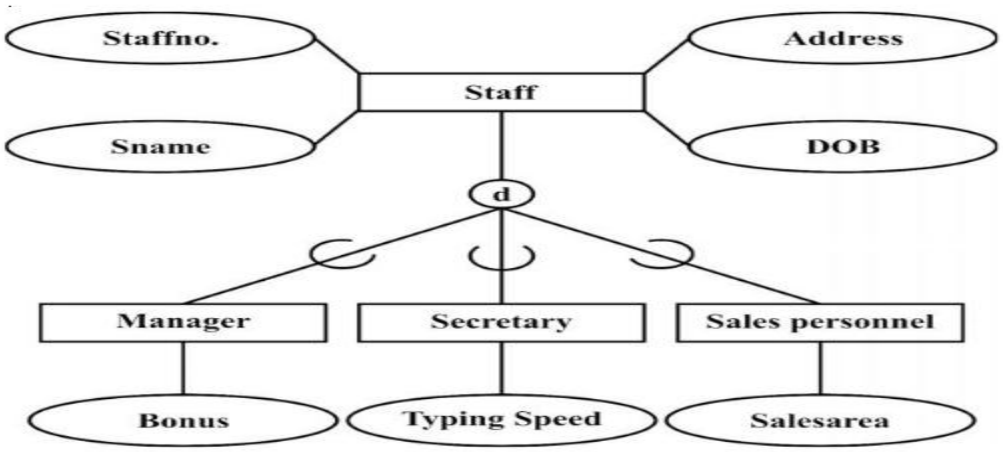
Type Inheritance

- The type of an entity is defined by the attributes it possesses, and the relationship types it participates in.

- Because an entity in a subclass represents the same entity from the super class, it should possess all the values for its attributes, as well as the attributes as a member of the super class.
- This means that an entity that is a member of a subclass inherits all the attributes of the entity as a member of the super class;
- as well, an entity inherits all the relationships in which the super class participates

Specialization

The process of defining a set of subclasses of super class
The specialization is a top down approach of super class and subclasses
The set of sub classes is based on some distinguishing characteristic of the super class.



Notation for Specialization

To represent a specialization, the subclasses that define a specialization are attached by lines to a circle that represents the specialization, and is connected to the super class.

The subset symbol (half-circle) is shown on each line connecting a subclass to a super class, indicates the direction of the super class/subclass relationship.

Attributes that only apply to the sub class are attached to the rectangle representing the subclass. They are called specific attributes.

A sub class can also participate in specific relationship types

Reasons for Specialization

Certain attributes may apply to some but not all entities of a super class. A subclass is defined in order to group the entities to which the attributes apply.

The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass.

Summary of Specialization

Allows for:

Defining set of subclasses of entity type.

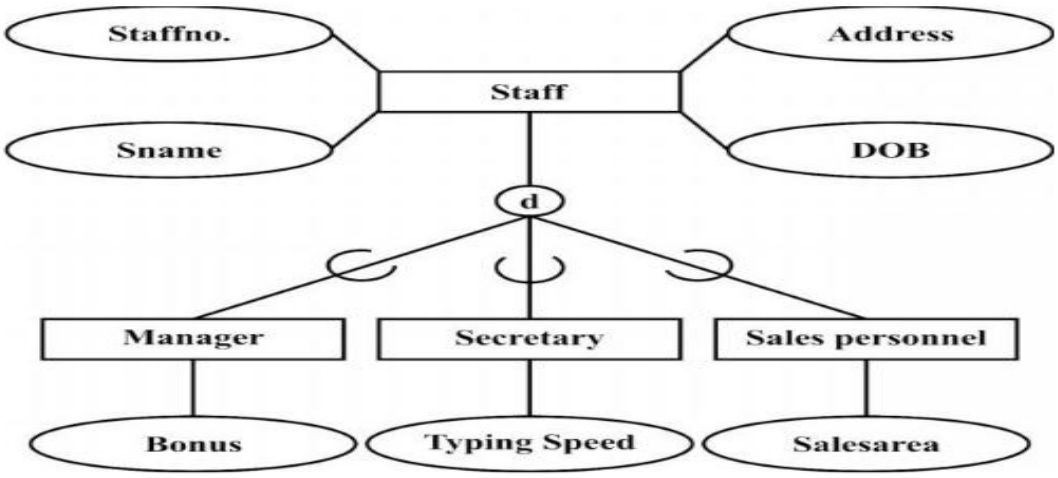
Create additional specific attributes for each sub class.

Create additional specific relationship types between each subclass and other entity types or other subclasses.

Generalization

Generalization is the reverse of specialization and this is a bottom-up approach

In Generalization, there are several classes with common features and generalizing into a super class.



Attribute Inheritance

- An entity in a Subclass may possess subclass specific attributes, as well as those associated with the Superclass

CODD'S RULE

Dr. E.F Codd was inventor of the relational database model.
This model say that whether the Database management system follow the relational model or not and what extents model is relational.
The article mentioned by Dr.E.F.Codd that according to these rule, There is no database management system fully implements all the 12 rules what he has been specified
In 1990, The codd rules extended 12 to 18 rules that's includes catalog, data types, authorization etc.

OverView of codd's rule

Sr.NO	Rule	Description
1	**The information rule:**	All the information in the database should be represented in the term of relational or table. Information should be stored as an values in a tables
2	**The guaranteed access rule**	All data must be accessible. The Rule say that there is fundamental requirement of primary key for each record in table ,and there should be no ambiguity by stating the table name and its primary key of the each record in the table along with columns name to be accessed.
3	**Systematic treatment of null values:**	Null values could not be treated as blank space or zero values, The null values are known as unknown values, unassigned values should be treated as missing Information and inapplicable information that should be treated as systematic , distinct from regular values
4	**Active online catalog based on the relational model:**	The system must support an online catalog based data dictionary which hold the information or description about the table in the database
5	**The comprehensive data sublanguage rule:**	The system must support at least one relational language that through which the data in the database must be Accessed . The language can be used both interactively and within application programs. The Language must Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval),security and integrity constraints, and transaction management operations (begin, commit, and rollback).
6	**The view updating rule:**	All the view must be theoretically updatable can be updated the system
7	**High-level insert, update, and delete:**	This rules states that in the relational model, the structured query language must performed data manipulation such as inserting ,updating and deleting record on sets of rows in the table
8	**Physical data independence:**	Any change made in the data is physically stored in term of data is stored in the file system through array and link list must not effect application that access the data structure
9	**Logical data independence:**	This rule state that changes in the logical level(rows, columns and so on) must not change to the application's structure
10	**Integrity independence:**	Data integrity constraints should be considered as separated from application program, the structured query language which defines data integrity constraint must be stored in the database in term of data in table that is, in the catalog and not in the application.

11	**Distribution independence:**	The rules states that the data can be stored centrally on the single machine or it can be stored in the various location(distributed) on various machine but it should be invisible to the user i.e. the user does not location of data is stored whether on the single machine(Centrally stored) or the distributed stored. If the location of database in change then the existing application must continually access the change database
12	**The non-subversion rule:**	The system must not have features that allow you to subvert database structure integrity. Basically, the system must not include back doors that let you cheat the system for features such as administrative privileges or data constraints.

Codd's rule in detail

1)The information rule:

I)All the information in the database should be represented in the term of relational or table. Information should be stored as an values in a tables

II)Data should be stored in form a table and no other means to stored the data

III)E.g. If want to stored data of student in the form of table.Consider name of Table is Students , it has four field(i.e column name) Roll_no, Firstname ,Lastname and date_of_birth and Consist five record man's Five rows

Students Table			
Roll_no	Firstname	Lastname	date_of_birth
101	Sachin	Godbole	17/07/1981
102	Mahavir	Jain	04/12/1985
103	Dinesh	Maheshwari	09/10/1987
104	Yogesh	Lad	06/11/1985
105	Mahesh	Thorat	07/06/1989

2) Guaranteed access rule

I) The guaranteed access

All data must be accessible. The

Rule say that there is fundamental requirement of primay key for each record in table ,and there should be no ambiguity by stating the table name and its primary key of the each record in the table along with columns name to be accessed rule

II)For accessing the data from the table , we must provides Table name , Primary key(ie Each unique value for each record(row) in the table) and other column names in the

table to be accessed

III**)**Considered the above Students table, if we want to find the First name , Lastname and date_of_birth of student whose Roll_no is 103

IV)Here , the Roll_no is the primary key for the Students Table,

This Roll_no Columns is distinct from all other columns, based upon the primary key, all the information present in the table must be guaranteed accessed

3) **Systematic Treatment of Null values**

I)Null values could not be treated as blank space or zero values, The null values are known as unknown values, unassigned values should be treated as missing information and inapplicable information that should be treated as systematic , distinct from regular values

II)Null values is very important concept is the database ,A null values must be represented as missing information in the table , it is not same as the blank space, dash, or zero, hash or any other representation

III)A null values means that we don't know what information must be provided or entered in to this field name

IV)Null values must be handled logically and consistent manner

4)**Active online catalog based on the relational model:**

I)The system must support an online catalog based data dictionary which hold the information or description about the table in the database

II) User Tables:

The user table contains the data about the table which is created by any users in the database systems

III)System tables: The system table contains the data about the structure of the database and data base object

IV)Metadata: The data which hold the description of table in the database, the table structure, database structure , the relationship among the tables, the queries and on , This data id often called as metadata , in short term, Metadata is data about the data

V)The collection of the system tables is known as the system catalogs or data dictionary

5) The comprehensive data sublanguage rule:

I)The system must support at least one relational language that through which the data in the database must be accessed

II) The language must support all the operation of the following items:

Data definition
View definition
Data manipulation
Integrity constraints
Authorization
Transaction boundaries (begin, commit and rollback)

6) The view updating rule:

I)All the view must be theoretically updatable can be updated by the system

II) There is ambiguity in this rule, the Structured query language support a single updation at a time suppose if we try combine two or more tables a for a complex views and try to update the views and the DBMS would fail to update the records to the respective tables, thereby violating this rule.

IV)If that view doesn't include the primary key columns in the view, then each record in the table cannot be updated, thereby violating this rule. Eg. If Roll_no column is not present in the view then it is not possible update the view of the student table

7) High-level insert, update, and delete:

I)This rules states that in the relational model, the structured query language must performed data manipulation such as inserting ,updating and deleting record on sets of rows in the table

II)You expected from the RDBMS, that you can retrieves all the record from table applying single command on the set of tables, or by using single query statement, this rules state that not only retrieves all the record from table but also you can apply the delete , insert, and update multiple records should possible by using the single command

III)Considered an example, if you want to delete the record of the invoices table which are older than six years, you don't have locate position each record and delete them individually , you should able to delete set of records in the table using One single command

IV)The same concept can be apply to inserting and updating the record

8)Physical data independence:

i) Any change made in the data is physically stored in term of data is stored in the file system through array and link list must not effect application that access the data structure

This rule say that any change is made in the back end(SQLServer/oracle) must not effect front end application(Visual basic/Java)

If the database file renamed or database location is change, then this should not have effect on the application.

9)Logical data independence:

This rule state that changes in the logical level(rows,columns and so on) must not change to the application's structure

This rule state that it should possible to change the database design or alter the database design without the user being aware of it.

These change could be to adding a new table in the datable or to delete the table from the database but the application must effect for accessing the data structure

Consider an example if want the performance search the record in the table, for that reason you have split the Customer table in to part i.e Customer_India and Customer_Rest,This allows to search a record in the Customer_India rapidly, but what about the

exiting user who is referring to the Customer table. In practice it can be done by creating a view which will combines two table into the single table with the same name. so that there should be effect on the application.

10)Integrity independence:

Data integrity constraint s should be considered as separated from application program, the structured query language which defines data integrity constraint must be stored in the

database in term of data in table that is, in the catalog and not in the application.

Referential integrity and entity integrity is integral part of the relational database , in more specific term, the following two integrity should be apply to the relational database.

i)Entity integrity: The column which have the primary key value should not contain missing values or duplicate value. This mean the column should contain the null values and unique

value in the each record set

ii)Referential integrity:

The column which have the foreign key value, there must exist a matching primary key column value mean the foreign key column have duplicate value must be referential to primary key column value.

11)Distribution independence

The rules states that the data can be stored centrally on the single machine or it can be stored in the various location(distributed) on various machine but it should be

invisible to the user i.e the user does not location of data is stored whether on the single machine(Centrally stored) or the distributed stored

.If the location of database in change then the existing application must continually access the change database

One of the important benefits of networking is that it allows multi-user access to a database; that is, the users can access the data which is distributed across the network.

However, it is also possible to distribute the data across the same network.

This rule also state that even if the table moves from one location to another location the user should aware of it, it should be transparent to the user, changing in the location

mean that the application should not be rewritten.

12)Non-subversion rule

The system must not have features that allow you to subvert database structure integrity. Basically, the system must not include back doors that let you cheat the system for features

such as administrative privileges or data constraints.

To understand another way, a user should not be allowed access the database by means of other way, other than SQL

CHAPTER IV: UML

UML or Unified Modeling Language is a specification which is used in the software engineering field. It can defined as a general purpose language which is used to design as graphical notation which is used an abstract model and this abstract model is used in the system. That system is called as UML or Unified Model language.

Why Modeling is required and what is the principle of Model?

Analysis the problem domain that is simply reality captures requirements in the design the model, visualize the system in its entirety, and specify the structure and / or behavior of the system choose your model well The choice of model such way that it should be through analysis of the problem and the design of the solution.

Every model in the system can be expressed at different levels of accuracy-the same model can be scaled up (or down) to different granularities.

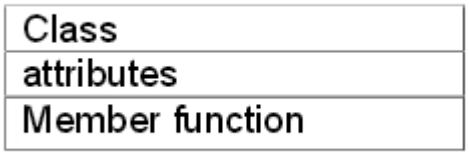
The best models in the system are closer to reality-simplify the model as much as possible and don't hide important details.

No single model suffices-every nontrivial system has different set of dimensions to the problem and has much solution UML is an modeling Language but not a methodology or process , the first concept is developed by Grady Booch , James Rumbaugh and Ivar Jacobson at Rational Software.

This model is accepted as a standard by the Object Management Group (OMG), in 1997

TYPE OF UML

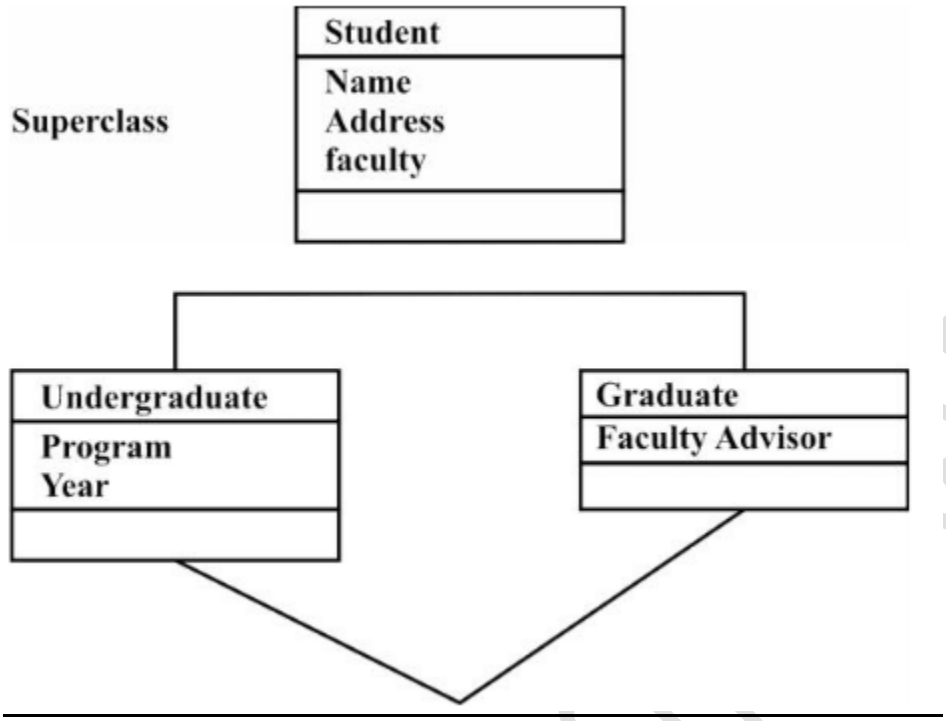
The Main purpose of the class diagram is include the classes within a model. In the object oriented programming, the classes has certain attributes(i.e. data member), operations(member function) and relationship among the objects , In the UML the class diagram can be include very easily. The fundamental part of the class diagram is the class icon which can represented a class. The class icon which is shown in the figure



A class icon is simply a rectangle divided into three compartments. The topmost compartment contains the name of the class. The middle compartment contains a list of attributes (member variables), and the bottom compartment contains a list of operations (member functions). In many diagrams, the bottom two compartments are omitted. Even when they are present, they typically do not show every attribute and operations. The goal is to show only those attributes and operations that are useful for the particular diagram.

If two classes are very similar it may be helpful to put the similarities into a more general class called a superclass.

For example, if you set up a superclass called Student, then Graduate Student and Undergraduate Student can be subclasses of Student.



USECASE DIAGRAM

A use case is a set of scenarios that shows an interaction between a user and a system.
A use case diagram shows the relationship among actors and use cases.
The two main components of a use case diagram are use cases and actors.

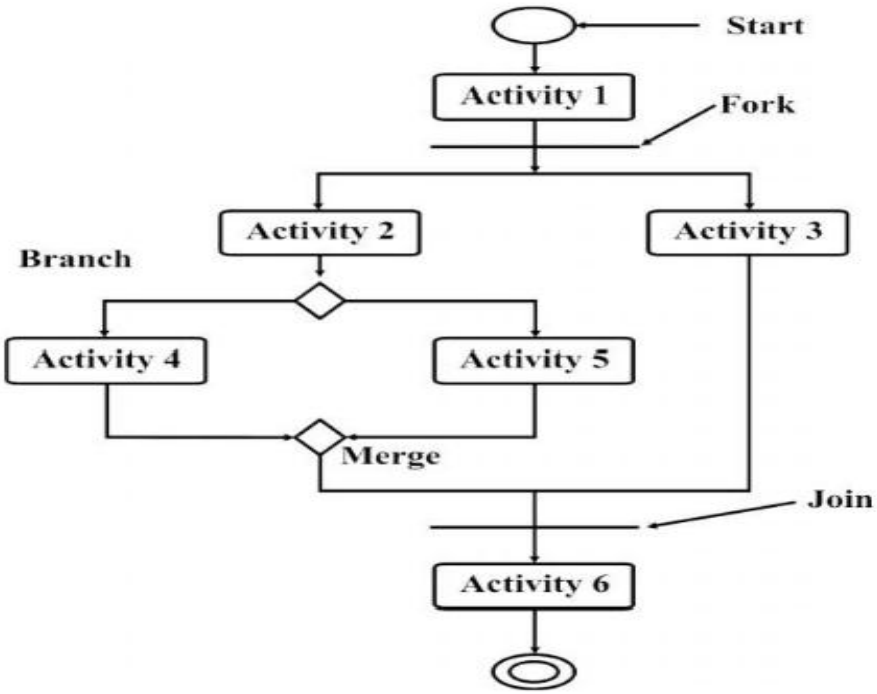
An actor is represents a user or another system that will interact with the system you are modeling.
A use case is an external view of the system that represents some action the user might perform in order to complete a task.

ACTIVITY DIAGRAMS

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency.
In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control. Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:
rounded rectangles: represent activities;

Diamonds: represent decisions;

Bars represent the start (split) or end (join) of concurrent activities;
A black circle represents the start (initial state) of the workflow;
An encircled black circle represents the end (final state).
Arrows run from the start towards the end and represent the order in which activities happen.
Hence they can be regarded as a form of flowchart
. Typical flowchart techniques lack constructs for expressing concurrency.
However, the join and split symbols in activity diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops.



SEQUENCE DIAGRAMS

Sequence diagrams is involved how to the object are interacted which are arranged in a time sequence. The Sequence Diagram which is use the flow of events to determine what objects and interactions I will need to accomplish the functionality specified by the flow of events.

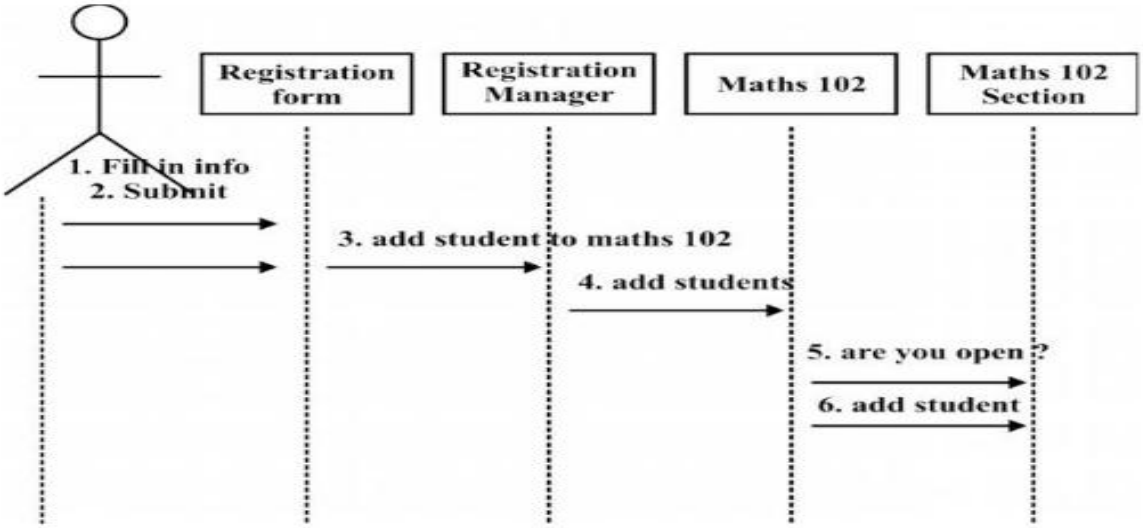
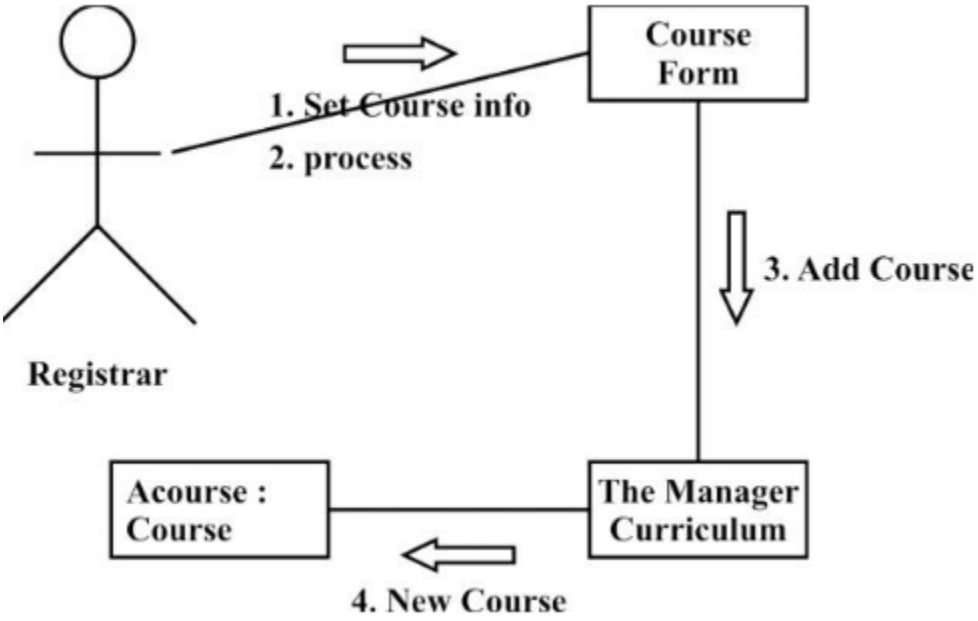


Figure shows how a student successfully gets added to a course. The student let's call him Mahesh) fills in some information and submits the form. The form then talks to the Manager and says "add Joe to Mahesh 102."
The manager tells Math 102 that it has to add a student.
Math 102 says to Section 1 "are you open?" In this case,
Section 1 replies that they are open, so Math 103 Tells section 1 to add this student. Again, sequence diagrams are great tools in the beginning because they show you and your customer step-by-step what has to happen.

COLLABORATION DIAGRAMS

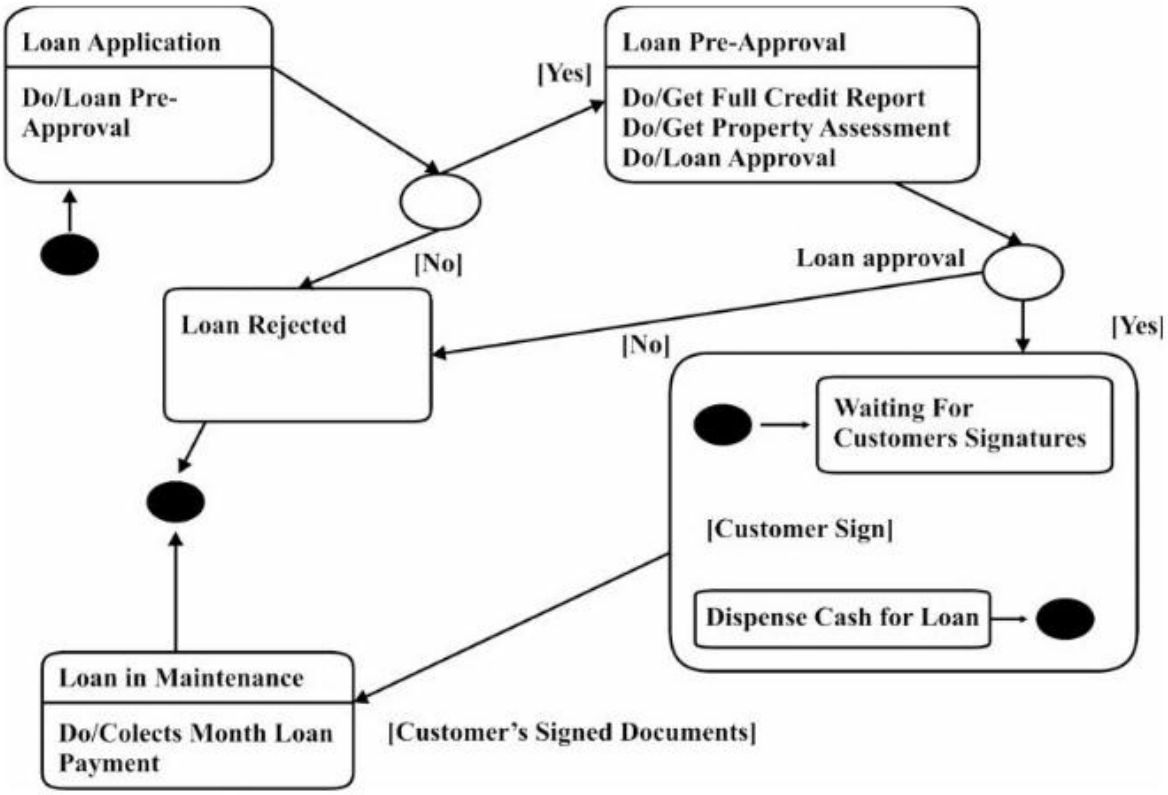


STATECHART DIAGRAM

The statechart diagram models the different states that a class can be in and how that class transitions from state to state. It can be argued that every class has a state, but that every class shouldn't have a state chart diagram. Only classes with "interesting" states--that is, classes with three or more potential states during system activity--should be modeled.

As shown in Figure 5, the notation set of the statechart diagram has five basic elements: the initial starting point, which is drawn using a solid circle; a transition between

states, which is drawn using a line with an open arrowhead; a state, which is drawn using a rectangle with rounded corners; a decision point, which is drawn as an open circle; and one or more termination points, which are drawn using a circle with a solid circle inside it. To draw a statechart diagram, begin with a starting point and a transition line pointing to the initial state of the class. Draw the states themselves anywhere on the diagram, and then simply connect them using the state transition lines.



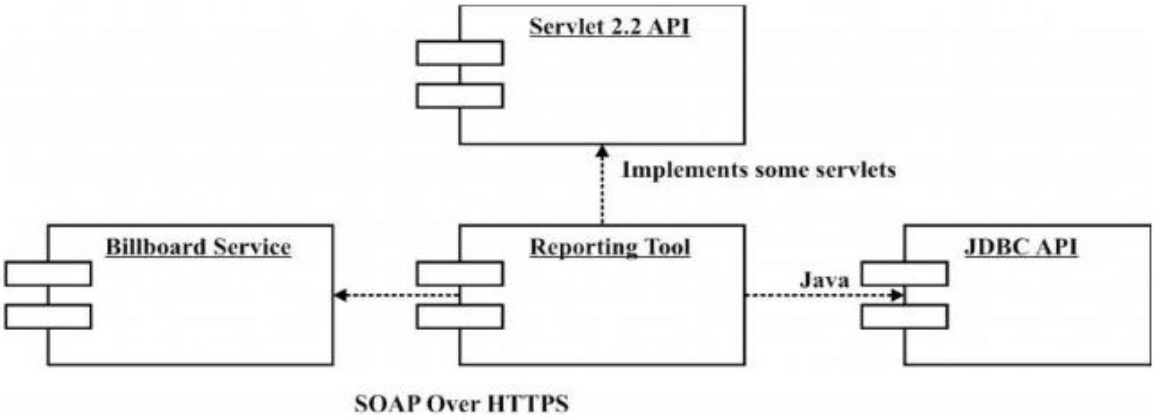
The example statechart diagram in Figure how some of the vital information they can communicate. For example, you can tell that loan processing department to begin in the Loan Application state. When the pre-approval process is over, depending whatever output comes, and then you move to either the Loan Pre-approved state or the Loan Rejected state. This decision, which is made during the transition process, is shown with a decision point—the empty circle in the transition line. By looking at the example, a person can tell that a loan cannot go from the Loan Pre-Approved state to the Loan in Maintenance state without going through the Loan Closing state. Also, by looking at our example diagram, a person can tell that all loans will end in either the Loan Rejected state or the Loan in Maintenance state.

COMPONENT DIAGRAM

A component diagram provides a physical view of the system. Its purpose is to show the dependencies that the software has on the other software components (e.g., software libraries) in the system. The diagram can be shown at a very high level, with just the large-grain components or it can be shown at the component package level.

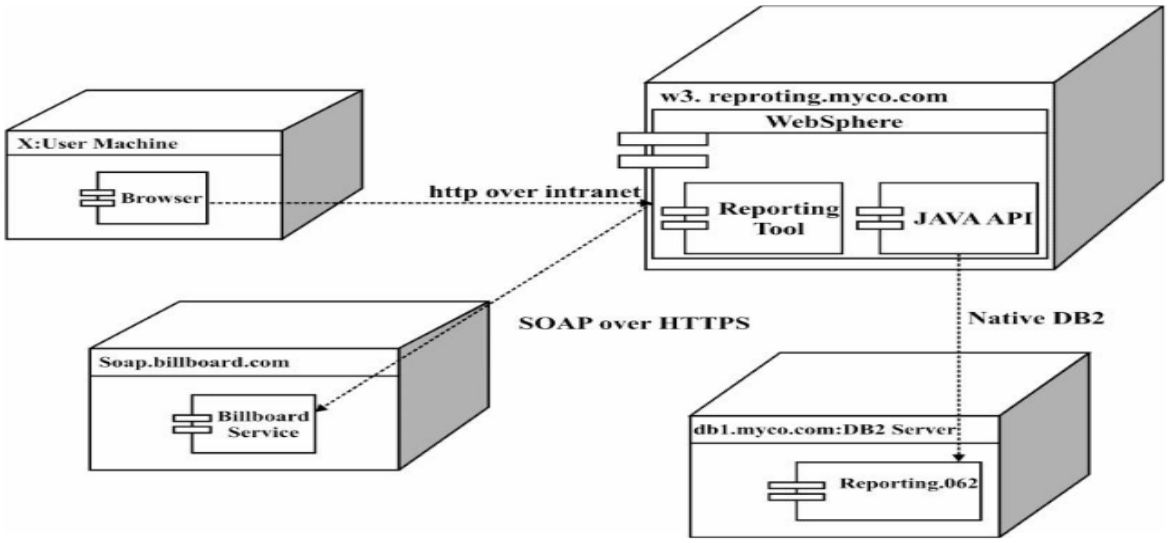
Modeling a component diagram is best described through an example.

Figure shows four components: Reporting Tool, Billboard Service, Servlet 2.2 API, and JDBC API. The arrowed lines from the Reporting Tool component to the Billboard Service, Servlet 2.2 API, and JDBC API components mean that the Reporting Tool is dependent on those three components.



DEPLOYMENT DIAGRAM

The deployment diagram shows how a system will be physically deployed in the hardware environment. Its purpose is to show where the different components of the system will physically run and how they will communicate with each other. Since the diagram models the physical runtime, a system's production staff will make considerable use of this diagram. The notation in a deployment diagram includes the notation elements used in a component diagram, with a couple of additions, including the concept of a node. A node represents either a physical machine or a virtual machine node (e.g., a mainframe node). To model a node, simply draw a three-dimensional cube with the name of the node at the top of the cube. Use the naming convention used in sequence diagrams: [instance name] : [instance type] (e.g.,"w3reporting.myco.com : Application Server").



The deployment diagram in Figure shows that the users access the Reporting Tool by using a browser running on their local machine and connecting via HTTP over their company's intranet to the Reporting Tool.

This tool physically runs on the Application Server named w3reporting.myco.com. The diagram shows the Reporting Tool component drawn inside of IBM Web Sphere, which in turn is drawn inside of the node w3.reporting.myco.com.

The Reporting Tool connects to its reporting database using the Java language to IBM DB2's JDBC interface, which then communicates to the actual DB2 database running on the server named db1.myco.com using native DB2 communication.

In addition to talking to the reporting database, the Report Tool component communicates via SOAP over HTTPS to the Billboard Service.

CHAPTER V: RELATIONAL DATABASE MODEL

RELATIONAL DATABASE MODEL:

E.F.codd first proposed the relational database Model also he is known as the father of Relational model.

Relation model was attempted to specify the database structure in term of matrix.ie the database should contain tables. The table is in form of set of Columns and Rows.

The relational model is set of 2 dimensional table consists of rows and columns.

Tables in the database is known as relation and Columns in the table is called as attributes of an tables and rows in the table is called records or tuples.

In the relational database model consists of set of tables having the unique name.

One row in the table represents a relationships among the another table in the database. The set value in the one table is related to the set of values in another table. Thus the table is represents a collection of relationship, the relationship among the tables in the form primary key–foreign key relationship.

LOGICAL VIEW OF DATA:

1 introduction

Logical structure of tables is consist of 2-dimensional tables consist of numbers of horizontal Rows and vertical columns

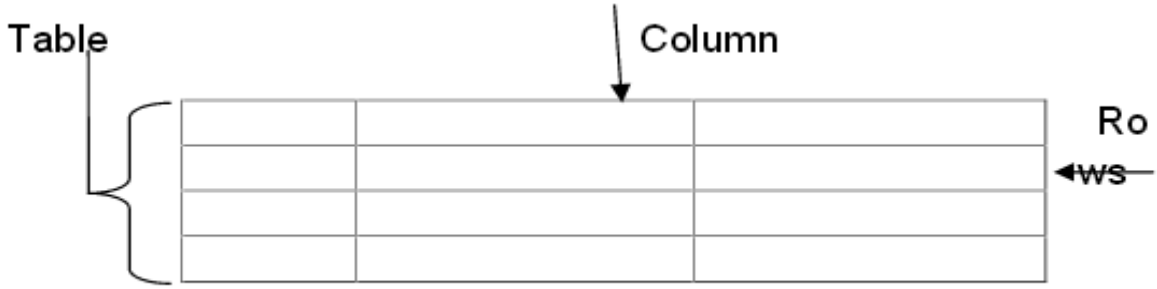


Table is an abstract entity which does not say how the data is stored in the physical memory of the computer system

Each table in the database has its own unique name trough which we can refer the content of the table by the unique name

2. Characteristics of an table

I) A tables in the database must be in the two-dimensional structure which consist number of rows and columns

II) Each row in table as called as record or tuple can represent as a single entity which is occur within the entity set i.e. Customer record in the Customer table.

III) Each column name in the table is called as attribute and each row in the table is called as record. Each column name in the table is unique name i.e. no duplicate name in the same table cannot be repeated.

IV) Each rows/ column interaction represent a single data item.

V) All the value in the column must be represent in the same data format.

VI) Each column has the specific range of values, and also refer as the domain attribute.

VII) The order of rows and columns is not limited to the DBMS.

3. Example

There is Customer Table contain all information about the Customer

Cust_id
Cust_Name
Cust_Age
Cust_Address
Cust_Mobile_No
Cust_Phone_No

Cust_id	Cust_ Name	Cust_ Age	Cust_ Address	Cust_Mobile _No	Cust_Phone_ No
1	Yogesh	20	Worli	9892456123	0224672345
2	Ramesh	23	Bandra	9320896742	0225678894
3	Ram	18	mahim	9819674534	0224678678
4	Pramod	24	Khar Road	9821673445	0223456478
5	Yatin	25	Dadar	9892396735	0222456783
6	Tushar	26	Matunga	9867458432	0226783452

Attribute

- Each column in the above table represent the data item in the database.
- Each column in the table represent the attribute in the table.
- At least one column consist in the table.
- There must be one unique column in the table , this means that no two columns has the same name in the same table ,it is possible to have two column with same column name but it in the different table.
- The ANSI/SQL Standard does not specify a maximum numbers of rows and columns in th e table.

Eg.Cust_id ,Cust_Name ,Cust_Age ,Cust_Address,Cust_Mobile_No, Cust_Phone_No are the attributes of the Customer Table

Records/Tuples

- A single Record consist all the information of the single entity.
- Each horizontal row in the Customer table represented a single entity.
- A Table consist any number of rows, The ANSI/SQL Standard Does not specify the limits of rows in the table.
- Empty table is called when there is zero row consist in the table

KEY

Definition

A Column value in the table that uniquely identifies a single record in the table is called key of an table.

A attribute or the set of attribute in the table that uniquely identifies each record in the entity set is called a key for that entity set

Types of keys

Simple Key: A key which has the single attribute is known as a simple key

Composite key: A key which consist two or more attributes is called a Composite Key.

Example:

Cust_id is a key attribute of Customer Table it is possible to have a single key for one customer i.e is Cust_id ie Cust_id =1 is only for the Cust_name =”Yogesh” please refer to the Customer Table which is mentioned above.

Types of key	Definition of Key
Super Key	A key is called super key which is sufficient to identify the unique record in the table
Candidate Key	A minimal super key is called Candidate key .A super key has no proper subset of candidate key
Primary Key	A candidate key is chosen as a principal to identify a unique
Secondary Key	
Foreign Key	an Column (or combination of Columns) in the one tables whose values is match the primary key in the another table

Types of key

1 Super Key

A key is called super key which is sufficient to identify the unique record in the table.

Customer Table

Cust_id	Cust_ Name	Cust_ Age	Cust_ Address	Cust_Mobile_ No	Cust_Phone_ No
1	Yogesh	20	Worli	9892456123	0224672345
2	Ramesh	23	Bandra	9320896742	0225678894
3	Ram	18	mahim	9819674534	0224678678
4	Pramod	24	Khar Road	9821673445	0223456478
5	Yatin	25	Dadar	9892396735	0222456783
6	Tushar	26	Matunga	9867458432	0226783452

Example

Here Cust_id attribute of the entity set Customer is uniquely identify Customer entity from another so The Cust_id is the Superkey. Another way is, the combination of Cust_id attribute and Cust_Name attribute is the Super key for the Customer Entity set.

Only the Cust_Name is not called the Super Key because several customer may have the Same Name

2. candidate key

Definition:

A minimal super key is called Candidate key .A super key has no proper subset of candidate key Here Minimum attribute of the super key is omitted unwanted attributed of an table that key is sufficient for identifying the unique record in the entity set so it is called as Candidate key.

The Candidate key is also known as the primary key.

Cust_id	Cust_ Name	Cust _Age	Cust_ Address	Cust_Mobile _No	Cust_Phone_ No
1	Yogesh	20	Worli	9892456123	0224672345
2	Ramesh	23	Bandra	9320896742	0225678894
3	Ram	18	mahim	9819674534	0224678678
4	Pramod	24	Khar Road	9821673445	0223456478
5	Yatin	25	Dadar	9892396735	0222456783
6	Tushar	26	Matunga	9867458432	0226783452

From above statement say combination of Cust_id attribute and Cust_Name is a super key for the Customer entity set it is required to distinguish one record on the Customer entity from another record of sane set.

But Cust_id attribute of the Customer entity is also known as minimal super key which also enough to distinguish one record from customer entity from another record from customer entity set,because Cust_Name is th additional attribute of the Customer table.

2 Primary key

Definition

Primary key of the table is a columns or combination of the some columns whose values is uniquely identify a single record in the table.

Primary key state no two record of the table contain the same value in that column or Combination of the column

It state that a unique identifier for the entity set.

Cust_id	Cust_ Name	Cust_ Age	Cust_ Address	Cust_Mobile _No	Cust_Phone_ No
1	Yogesh	20	Worli	9892456123	0224672345
2	Ramesh	23	Bandra	9320896742	0225678894
3	Ram	18	mahim	9819674534	0224678678
4	Ram	20	Khar Road	9821673445	0223456478
5	Yatin	25	Dadar	9892396735	0222456783
6	Tushar	26	Matunga	9867458432	0226783452

In above table the Customer Age cannot act as primary key hence the customer age column contain repeated values and

Customer Name also cannot act as primary key because it earlier state that several customer may have the same name hence

Cust_Name column has the repeated values .

Hence there Cust_id can act as the primary key in the Customer table this is only column which contain a unique set of values.

3 Secondary key

Definition

Secondary key of the table consist the column and combination of the some columns which meant for data retrieval purpose.

The secondary key not always required to primary key,
Other than the primary key there are some attribute which is required to retrieve data from the customer table using the another attribute
such as Cust_Name and Cust_Age columns

Cust_id	Cust_ Name	Cust_ Age	Cust_ Address	Cust_Mobile_ No	Cust_Phone_ No
1	Yogesh	20	Worli	9892456123	0224672345
2	Ramesh	23	Bandra	9320896742	0225678894
3	Ram	18	mahim	9819674534	0224678678
4	Ram	20	Khar Road	9821673445	0223456478
5	Yatin	25	Dadar	9892396735	0222456783
6	Tushar	26	Matunga	9867458432	0226783452

In the above Customer Table
Cust_Name and Cust_Age attribute act as the Secondary key

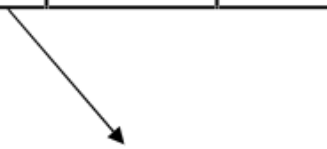
Foreign Key

A Column (or combination of Columns) in the one tables whose values is match the primary key in the another table is called as a foreign key.
Foreign key can also have one or more column like as primary key.
A single table may contain more than one foreign key which is related to the more than one table, the table which used the foreign key is said the referential integrity.

What the referential integrity

Referential integrity say the column which contain foreign key in one table must be primary key of another table
In general term, Foreign key of Table A must be Primary key of Table B
Example
Customer Table

Cust_id	Cust_ Name	Cust_ Age	Cust_ Address	Cust_Mobile_ No	Cust_Phone_ No



Account Table

Account No	Cust_id	Account type	Balance	Description

In The above example Cust_id is the primary key for the customer Table while Cust_id is the foreign key for the Account table
Here the data type assigned to column and Number of column in the foreign key is same as to the primary key.

Cust_id	Cust_ Name	Cust_ Age	Cust_ Address	Cust_Mobile_ No	Cust_Phone_ No
1	Yogesh	20	Worli	9892456123	0224672345
2	Ramesh	23	Bandra	9320896742	0225678894
3	Ram	18	mahim	9819674534	0224678678
4	Ram	20	Khar Road	9821673445	0223456478
5	Yatin	25	Dadar	9892396735	0222456783
6	Tushar	26	Matunga	9867458432	0226783452

Account No	Cust_id	Account type	Balance	Description
101	1	Saving	10,000	
102	2	saving	20,200	
103	2	Saving	20,200	
104	3	Current	11,000	
105	4	Saving	50,000	

REALATIONAL INTEGRITY RULES

1) Entity Integrity

Entity Integrity ensure that there is no duplicate records in the table and each field that recognizes each record in the table must have unique value and not having null values

Entity Integrity specifies that every instance of entity have the unique values i.e. primary key must be kept and must have the values other than null values.

Entity Integrity is the mechanism the Database management system provides to maintain primary keys. The primary key is known as unique identifier for each rows in the table. Entity Integrity must have two properties for primary keys:

♣The primary key must be unique for each row in the table that is no two primary key having the same value in the same table,

The primary key values must be distinct i.e. the value could not be repeated.

♣The primary key values should not contain null values, primary key must be NOT NULL

The uniqueness property ensures that the primary key of each row uniquely identifies it; there are no duplicates. The second property ensures that the primary key has meaning, has a value; no component of the key is missing.

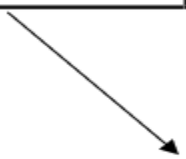
2. Referential Integrity

Referential integrity is a property of data which, when satisfied, requires every value of one attribute (column) of a relation(table) to exist as a value of another attribute in a different (or the same) relation (table).

For referential integrity to hold in a relational database, any field in a table that is declared a foreign key can contain only values from a parent table's
primary key or a candidate key. For instance, deleting a record that contains a value referred to by a foreign key in another table would break referential integrity.
Some relational database management systems(RDBMS) can enforce referential integrity, normally either by deleting the foreign key rows as well to maintain integrity, or by returning an error and not performing the delete.
Foreign key
A column or collection of column in one table whose values must match the primary key in the other table is known as a foreign key

Cust_id	Cust_ Name	Cust _Age	Cust_ Address	Cust_Mobile_ No	Cust_Phone_ No
1	Yogesh	20	Worli	9892456123	0224672345
2	Ramesh	23	Bandra	9320896742	0225678894
3	Ram	18	mahim	9819674534	0224678678
4	Ram	20	Khar Road	9821673445	0223456478
5	Yatin	25	Dadar	9892396735	0222456783
6	Tushar	26	Matunga	9867458432	0226783452

Account No	Cust_id	Account type	Balance	Description
101	1	Saving	10,000	
102	2	saving	20,200	
103	2	Saving	20,200	
104	3	Current	11,000	
105	4	Saving	50,000	



In above example
Cust_id column of Account Table is foreign key for the Account table while it is primary key for the Customer Table

3. Other integrity rules

NOT NULL. As the integrity rules states column which specify the
NOT NULL values mean these column must contain some values which should not contain any NULL values
Unique. In this rules no two record or tuples have same values for the same attribute Check.
In this rule we can apply own integrity rules by applying CHECK Constraint.

CHAPTER VI: DATABASE NORMALIZATION

RELATIONAL DATABASE DESIGN PROCESS

The Relational Database model was proposed by E.F.Codd in 1969.The Relational Database Model is based on branch of mathematics called set theory and predicate logic. The idea behind to design the Relational Database model is that the database consist of series of unordered table or relation that can be manipulated using non-procedural process that return tables

Note:

it is Commonly thought that word relational in the relational model comes from the fact that the tables is related together in the relational model, but it is inconvenient way to think of the term , but it is not accurate.. The table which codd is specifies while in writings was actually referred to as relation (a related set of Information).

While designing relational database model you have consider in the mind that how choose a best model in the real world and how this best model is fitted in the database.

While designing the relational model you have to consider that which table you want to create, what column the table will consist, consider the relationship between the tables.

While developing the relational model it would be nice you process was totally clear and intuitive or it can be even better to automated.

- The benefits of a relational Database Design process.
- Data entry, updating and deleting would be efficient and simple in manner
- Data retrieval, summarization and reporting will be efficient
- Database must follows a well-designed model hence it behave predictably
- Large amount of information must stored in the database rather than in the application, the database must somewhat well documented
- Change to database structured are easy to make e.r creating database, tables , views.

Feature of Good Relational Database Design-Normalization

i)In the Relational Database Design, the process of organizing data to minimizing redundancy is known as Normalization

ii)The main aim of the Normalization is to decompose complex relation into smaller, well-structured relation

iii)Normalization is the process that involves dividing a large table into smaller table(which contain less redundant data)and stating the relationship among the tables.

iv)Data normalization or Database Normalization is also canonical synthesis is mean for preventing the inconsistent in a set of data by using unique values to reference common information

v)The main objective of the normalization is to isolate the data so that user can apply the operation such as addition, deletion and modification of a field in one table and then its propagated to the rest of the database through the well-defined relationships

vi)The same set of data is repeated in multiple tables of database so there are chances that data in the database may lead to be inconsistent, so while updating , deleting or inserting the data into the inconsistent database which leads to problem of data integrity

vii)If we can apply the normalization on the table we can reduce the problem of data inconsistency for some extent

Definition of Normalization

In the Relational Database Design, the process of organizing data to minimizing redundancy is known as Normalization

Main aim of the Normalization

1.Ensure data integrity

i)The correct data should be stored in the database

ii)This can be achieved by applying integrity rules in the database

iii)Integrity rules prevent duplicate values in the database

2.Prevent Data Redundancy in database

i)Non-Normalized data is more vulnerable to data anomalies. The same set of information is present in the multiple rows, now if we applying the updating rule on the

table then it lead to logical inconsistence this is known as update anomaly

An insufficiently normalized table might have one or more of the following characteristics:

update anomaly

The same set of information is present in the multiple rows, now if we applying the updating rule on the table then it lead to logical inconsistence. Consider an example of customer Table which contain set of attributes such as Cust_id ,Cust_Name,

Cust_Address,

Faculty_ID	Faculty_ Name	Faculty_Hire_ Date	Course_Code
386	Mahesh Lad	10/06/1994	ENG-207
197	Jayesh Shinde	12/06/1987	PP-205
197	Jayesh Shinde	12/06/1987	PP-206
234	Pramod Bhave	11/07/2005	?

Thus a change of address of a particular Customer will need update to multiple records. If the update is not carried out successfully—if, that is, the Customer's address is updated on some records but not others— then the table is remains in an inconsistent state. Specifically, the table provides conflicting answers to the question of what this particular customer's address is. This Known is known as an update anomaly.

The above Customer Table is Cust_id =567 having different address in the multiple records

An insertion anomaly There are some circumstances in which certain fact cannot recorded at all

Example Consider a table Faculty and Course_code consist the Column name

Faculty_ID,Faculty_Name,Faculty_Hire_Date,Course_Code

Faculty_ID	Faculty_ Name	Faculty_Hire_ Date	Course_Code
386	Mahesh Lad	10/06/1994	ENG-207
197	Jayesh Shinde	12/06/1987	PP-205
197	Jayesh Shinde	12/06/1987	PP-206
234	Pramod Bhave	11/07/2005	?

Thus we can add the record the details of any faculty member who teaches at least one course, but we cannot record the details of a newly- hired faculty member who has not yet been assigned to teach any courses except by setting the Course Code to null. This known as an insertion anomaly.

In the above Table Until the new faculty member, Pramod Bhave , is assigned to teach at least one course, his details cannot be recorded.

An deletion anomaly.

There are circumstances in which the deletion of data representing certain facts necessitates the deletion of some unrelated data . The "Faculty and Courses" table suffers from

This type of anomaly, for if a faculty member temporarily ceases to be assigned to any courses, we must delete the last of the records on which that faculty member appears, effectively also deleting the faculty member. This is known as a deletion anomaly.

Faculty_ ID	Faculty_Name	Faculty_Hire_ Date	Course_Code
386	Mahesh Lad	10/06/1994	ENG-207
197	Jayesh Shinde	12/06/1987	PP-205
197	Jayesh Shinde	12/06/1987	PP-206

All information about Mahesh Lad is lost when he temporarily ceases to be assigned to any courses.

Q. Explain Advantages and Disadvantages of Normalization.

Advantage of Normalization

1) Avoids data modification (INSERT/DELETE/UPDATE) anomalies as each data item lives in One place
2) Greater flexibility in getting the expected data in atomic granular.
3) Normalization is conceptually cleaner and easier to maintain and change as your needs change.
4) Fewer null values and less opportunity for inconsistency.
5) A better handle on database security.
6) Increased storage efficiency.
7) The normalization process helps maximize the use of clustered indexes, which is the most powerful and useful type of index available. As more data is separated into multiple tables because of normalization, the more clustered indexes become available to help speed up data access.

Disadvantage of Normalization

1)Requires much more CPU, memory, and I/O to process thus normalized data gives reduced database performance.
2) Requires more joins to get the desired result. A poorly-written query can bring the database down.
3) Maintenance overhead. The higher the level of normalization, the greater the number of tables in the database.

NORMAL FORM

Normal form are designed for addressing potential problem in the database such that inconsistent and redundant data which is stored in the database Normal form is based on relation rather than table . The normal form has a set of attribute which table should be satisfy.

The Following attributes are

1) They describe one entity.
2) They do not have duplicate rows, hence there must a primary key for each row.
3) The columns are unordered.
4) The rows are unordered.

Types of Normal Forms

1)Edgar F. Codd, the inventor of the relational model, introduced the concept of normalization and what we now know as the First Normal Form (1NF) in 1970.
2)Second Normal Form (2NF) and Third Normal Form (3NF) in 1971,
3)Codd and Raymond F. Boyce defined the Boyce-Codd Normal Form (BCNF) in 1974.
4)Fourth normal form(4NF)
5)Fifth Normal form(5NF)
6)Higher normal forms were defined by other theorists in subsequent years, the most recent being the Sixth Normal Form (6NF) introduced by Chris Date, Hugh Darwen, and Nikos Lorentzos in 2002.

First Normal Form

This Normal form is introduced by Edgar F. Codd , is Known as First Normal Form(1NF) in 1971

Definition

A relational database table which consist first normal form (1NF) is to meets certain minimum set of criteria. These criteria are basically concerned with ensuring that the table is a faithful representation of a relation and that it is free of repeating groups

1. There are no duplicated rows in the table.
2. Each cell is single-valued (i.e., there are no repeating groups or arrays).
3. Entries in a column (attribute, field) are of the same kind.

Let us consider the example

Consider a table"Customer_Rental " consisting the attribute such as Customer_NO , Cust_Name Property_no,P_Address, Rent_start, Rent_finish,Rent ,Owner_No,Owner_Name

Customer_ NO	Cust_Name	Property_ no	P_Address	Rent_start	Rent_finish	Rent	Owner_ No	Owner_Name
CR78	Mahesh Lad	PG34 PG78	Nerul,Navi Mumbai Turbhe, Navi mumbai	1-July-91 1-Nov-95	30-Oct-95 1-Nov-98	450 500	C045 C093	Sanjay More Mahavir Jain
CR98	Pramod Patel	PG34 PG36 PG78	Nerul,Navi Mumbai Kalyan,Thane Karjat,Raigad	1-July-95 1-Nov-97 1-july-96	30-Oct-98 1-Nov-99 1-Nov-97	450 350 450	C045 C093 C093	Sanjay More Mahavir Jain Mahavir Jain

The above table does not contain the atomaic values in the Property_no, P_Address , Rent_start, Rent_finish,Rent,Owner_No,Owner_Name Hence it is called un-normalizes table, we cannot Insert ,update and delete the record from the table because it is inconsistent state .The above table has to be normalized

Customer_ NO	Cust_Name	Property_ no	P_Address	Rent_start	Rent_finish	Rent	Owner_No	Owner_Name
CR78	Mahesh Lad	PG34	Nerul,Navi Mumbai	1-July-91	30-Oct-95	450	C045	Sanjay More
CR78	Mahesh Lad	PG78	Nerul,Navi Mumbai	1-Nov-95	1-Nov-98	500	C093	Mahavir Jain
CR98	Pramod Patel	PG34	Nerul,Navi Mumbai	1-July-95	30-Oct-98	450	C045	Sanjay More
CR98	Pramod Patel	PG36	Kalyan,Thane	1-Nov-97	1-Nov-99	350	C093	Mahavir Jain
CR98	Pramod Patel	PG78	Karjat,Raigad	1-july-96	1-Nov-97	450	C093	Mahavir Jain

The above table show the same set of data as the previous table however we have eliminated the repeated groups.so the table shown in the above table to be in First Normal form(1NF)

Second Normal Form

Second Normal Form based on the concept of Full Functional Dependency and it tries remove the problem of redundant data in the First normal form.

Definition: A 2NF relation in 1NF and every non-primary key attribute is fully functionally dependent on the primary key

Converting from 1NF to 2NF:

- o Firstly Identify the primary key for the 1NF relation.
- o Identify whether the functional dependencies in the relation.

o If partial dependencies exist on the primary key remove them by placing then in a new relation along with a copy of their determinant.

Example

Functional Dependency for Customer_Rental Relation

Step 1 :Primary key: Customer_No + Property_n

step 2 :Full Functional Dependency:(Customer_No+Property_No)->(Rent_Start, Rent_Finish)

Step3Partial Dependency:(Customer_No+Property_No)->Cust_Name(Customer_No+Property_No)->(P_Address, Rent,Owner_No,Owner_Name)

Customer_NO	Cust_Name	Property_no	P_Address	Rent_start	Rent_finish	Rent	Owner_No	Owner_Name

Customer Relation

Customer_NO	Cust_Name
CR78	Mahesh Lad
CR98	Pramod Patel

Rental Relation

Customer_NO	Property_No	Rent_start	Rent_finish
CR78	PG34	1-July-91	30-Oct-95
CR78	PG78	1-Nov-95	1-Nov-98
CR98	PG34	1-July-95	30-Oct-98
CR98	PG36	1-Nov-97	1-Nov-99
CR98	PG78	1-july-96	1-Nov-97

Property_owner Relation

Property_No	P_Address	Rent	Owner_No	Owner_Name
PG34	Nerul,Navi Mumbai	450	C045	Sanjay More
PG78	Nerul,Navi Mumbai	500	C093	Mahavir Jain
PG36	Kalyan, Thane	350	C093	Mahavir Jain

Here Customer_no is the only key to identify The Customer name hence Customer_No is the primary key in the Customer

Relation Table but Foreign key in the Rental relation table.

Third Normal Form

Third Normal form Based on the concept of transitive dependency.

A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key.

Converting from 2NF to 3NF:

o Identify the primary key in the 2NF relation.
o Identify functional dependencies in the relation.
o If transitive dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their dominant.

Property_Owner to 3NF Relations

Property_owner Relation

Property_No	P_Address	Rent	Owner_No	Owner_Name

Transitive Dependency:

(Customer_No+Property_No)->Owner_No

Owner_No ->OName

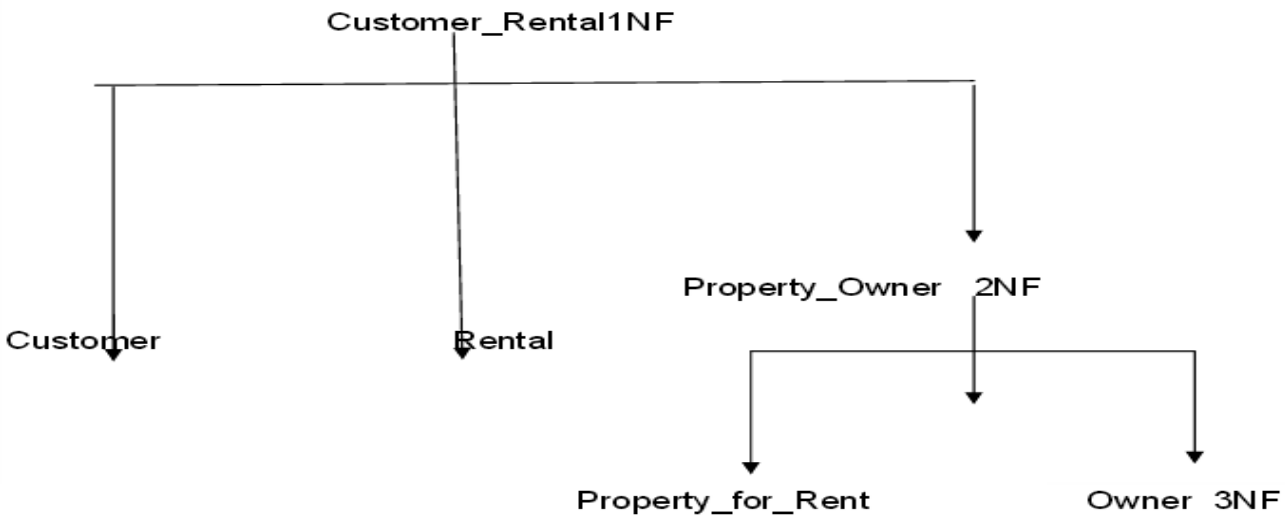
Property_for_Rent

Property_No	P_Address	Rent	Owner_No
PG34	Nerul,Navi Mumbai	450	C045
PG78	Nerul,Navi Mumbai	500	C093
PG36	Kalyan,Thane	350	C093

Owner

Owner_No	Owner_Name
C045	Sanjay More
C093	Mahavir Jain

Process of Decomposition



Boyce-Codd Normal Form (BCNF)

o Based on functional dependencies that takes into account all candidate keys in a relation.
o For a relation with only one candidate key, 3NF and BCNF are equivalent.
o A relation is in BCNF, if and only if every determinant is a candidate key.
o Violation of BCNF may occur in a relation that–contains 2 (or more) composite keys–which Overlap and share at least 1 attribute 3NF to BCNF.
o Identify all candidate keys in the relation.
o Identify all functional dependencies in the relation.
o If functional dependencies exists in the relation where their determinants are not candidate keys for the relation, remove the functional dependencies by placing them in a new relation along with a copy of their determinant.

Example- 3NF to BCNF Relations
Client_Interview Relation

Client_No	Interview_Date	Interview_Date	Staff_No	Room_No
CR76	13/05/98	10.30	SG5	G101
CR56	13/05/98	12.30	SG5	G101
CR74	13/05/98	12.30	SG37	G102
CR56	01/06/08	10.30	SG5	G102

(Client_No, Interview_Date) -> (Interview_Time, Staff_No, Room_No)

(Staff_No, Interview_Date, Interview_Time) -> Client_No

(Room_No, Interview_date, Interview_Time) -> Staff_No, Client_No

(Staff_No, Interview_Date) -> Room_No

Client_No	Interview_Date	Interview_Date	Staff_No
CR76	13/05/98	10.30	SG5
CR56	13/05/98	12.30	SG5
CR74	13/05/98	12.30	SG37
CR56	01/06/08	10.30	SG5

Staff_No	Interview_Date	Room_No
SG5	13/05/98	G101
SG37	13/05/98	G102
SG5	01/06/08	G102

CHAPTER VII: RELATIONAL ALGEBRA AND CALCULUS

DATA MANIPULATION LANGUAGES

In order to make the database more useful, then it should possible to store information in database or retrieve the information from the database. This important role is perform by database Manipulation Language

There are two types of Data Manipulation language

-Navigational (Procedural)

•The query specifies (to some extent) the strategy use to find the desired result eg relational algebra

-Non-navigational (non-procedural)

•The query only specifies what data is wanted, not how to find it e.g. relational calculus.

•Codd proposed a number of algebraic operations for the relational database model.

•In the Relation algebra there are two type of operation one is Unary operation and second one Binary operation

•Unary operation takes as input a single table and produces an output another table.

•Binary operations take as input two tables and produce as output another table.

Fundamental operation

- Unary operation
  - Projection operation(π)
    - Select Operation(σ)
    - Rename Operation(ρ)
  - Binary Operation
    - SET operation
      - Union operation(∪)
      - Difference Operation(-)
      - Intersection Operation(∩)
    - Join Operation(⋈)
    - Cartesian Product Operation(X)
    - Division Operation(%)

Selection operation

The Selection operator select the row from the table that satisfy a given predicate. This operation allows to manipulate data in the single relation. The Selection operation is defined by the symbol called sigma(σ). The predicate is appear at subscript of Sigma symbol(σ). The argument relation is present in the parenthesis after the σ

Syntax

σ<predicate><Comparsion_operator><Constant_value>(<input_table_name>)

Where Predicate: Name of the column in the table

Comparsion_Operator:=,<,<=,>,=>,<>

Example

Select all the student from the student table who's Roll no is greater than 300

Student

Roll No	Students_Name	Students_Address
134	Mary	3 Curry Road
356	John	4 Dockyard
500	Steve	6 Nepean Sea Road

$\sigma_{RollNo > 300}(Student)$

Roll No	Students_Name	Students_Address
356	John	4 Dockyard
500	Steve	6 Nepean Sea Road

We can combine several predicates into a larger predicate by using the connectives and (∧), or (∨), and not (¬).
To find the tuple in the student table where Student name is john and roll no is greater than 300
σ Students_Name="John"∧RollNo > 300(Student)

Roll No	Students_Name	Students_Address
356	John	4 Dockyard

Projection Operation(π)
This operator is used to select some of the attributes from the table to produces a desired result set
Note that Projection operation is used to eliminates the duplicates records in the table
Syntax
Π <attributes>(<Input_Table_Name>)
Example
1)Find the all record from the Stu dent table
Π Roll No,Students_Name, Students_Address(Student)

$\prod_{Roll\ No,\ Students\_Name,\ Students\_Address}(Student)$

Roll No	Students_Name	Students_Address
134	Mary	3 Curry Road
356	John	4 Dockyard
500	Steve	6 Nepean Sea Road

2)Find the Roll no, student name and student address whose rollno is greater than 300
Π Roll No,Students_Name, Students_Address(σ RollNo>300 (Student))

Rename Operator(ρ)
Rename operation gave alternate name to the given column or to any table by using the operator called Rename operator

This operator is used for selecting some specific column from multiple table(set of two or more tables) containing multiple columns having same column name
Rename operator is denoted by the greek letter rho(ρ)
Syntax
ρ<New Name for Column>(<Input_Table_Name>)
1)Find the all record from the Student table

$$\Pi \quad e.Roll\ No,\ e.Students\_Name,\ e.Students\_Address\ (\rho_e(Student))$$

e.Roll No	e.Students_Name	e.Students_Address
134	Mary	3 Curry Road
356	John	4 Dockyard
500	Steve	6 Nepean Sea Road

BINARY OPERATIONS
•Two relations are (union) compatible if they have the same set of attributes.
•Example, one table may represent suppliers in one country, while another table with same schema represents suppliers in another country.
•For the union, intersection and set-difference operations, the relations must be compatible.
Union, Intersection, Set-difference
•R1∪ R2
The union is the table comprised of all tuples in R1 or R2.
•R1∩R2
The intersection is the table comprised of all tuples in R1 and R2
•R1-R2
–The set-difference between R1 and R2 is the table consisting of all tuples in R1 but not in R2.

Union Operator
Union operator is used combine all the result form the first query to the result from the second query
Union operator does not eliminate duplicate record from the database and they prints the result expression
Syntax
(Relation1) ∪ (Relation 2)
Example
1)
Employee table

EMPNO	DEPTNAME	EMPFIRSTNAME	EMPLASTNAME
101	Sales	Jayesh	Shinde
102	R&D	Preetesh	Shinde
103	Marketing	Ganesh	lad
104	Sales	Pooja	Lad

2) Project Table

PROJECTNO	DEPTNAME	EMPNO
P1	R&D	103
P2	Sales	104
P3	HR	105

Union of the Two Table result in Employee Table and Project Table
Syntax : select deptname from Employee union
select deptname from Project;
Result:
The return value would be sales, marketing, R&D and HR.

Intersect Operator
This operator is find out all the tuples that all the Common to the result of Relation 1 and in the Result of Relation 2
Intersect operator does not eliminate duplicate record from the database and they prints the result expression
Syntax
•R1∩R2
Example: Get all the employee's full name that are working on a project.
Syntax: select EMPFIRSTNAME,EMPLASTNAME from Employee, Project where Project. EMPNO=Employee. EMPNO;
Result : Ganesh lad Pooja Lad

Difference Operation
The difference builds a relation consisting of all tuples appearing in the first and not the second of two specified relations.
The difference between two relation R1 and R2, R1 MINUS R2, is the set of all tuples belonging to R1 and not to R2.
Syntax R1-R2
Example: Find the employee that are in sales department and are not on project P2.
7
Syntax: select EMPNO from Employee where DEPTNAME='Sales' minus select EMPNO from Project where PROJECTNO='P2';Result:EMPNO=101
CARTESIAN PRODUCT
•R1×R2–
The Cartesian product is the table consisting of all tuples formed by concatenating each tuple inR1 with a tuple in R2, for all tuples in R2.
•The Cartesian Product is also an operator which works on two sets. It is sometimes called the CROSS PRODUCT or CROSS JOIN.
•It combines the tuples of one relation with all the tuples of the other relation.
Example of a Cartesian Product

R1

A	B
1	x
2	y

R2

C	D
a	s
b	t
c	u

R1XR2

A	B	C	D
1	x	a	s
1	x	b	t
1	x	c	u
2	y	a	s
2	y	b	t
2	y	c	u

Example
Employee Table

Empno	Empname	Deptno
101	Ramesh	100
102	Suresh	200
103	Rajesh	100

Department table

Deptno	Deptname
100	Sales
200	R &D

When we Join the Two table cross product Π e.Empno,e.Empname,e.Deptno,d.Deptno,d.Deptname (ρ e(Employee) X ρ e(Department))

e.Empno	e.Empname	e.Deptno	d.Deptno	d.Deptname
101	Ramesh	100	100	Sales
101	Ramesh	100	200	R &D
102	Ramesh	200	100	Sales
102	Ramesh	200	200	R &D
103	Rajesh	101	100	Sales
103	Rajesh	101	200	R &D

Join Operator

Join operator is used to retrieve data from multiple table or relations

Syntax

<tablename>

<tablename>

There are various types of join in relational algebra

Natural Joins–

Assume R1 and R2 have attributes A in common. Natural join is formed by concatenating all tuples from R1 and R2 with same values for A, and dropping the occurrences of A in R2–

R1R2 =Π A'(σ C(R1×R2))

–

where C is the condition that the values for R1 and R2 are the same for all attributes in A and A' is all attributes in R1 and R2 apart from the occurrences of A in R2.

Course Table

CourseId	Title	eid
CS51T	DBMS	123
CS52S	OS	345
CS52T	Networking	345
CS51S	ES	456

Instructor Table

eid	ename
123	Rao
345	Allen
456	Mansingh

Course ⋈ Instructor

Courseld	Title	eid	ename
CS51T	DBMS	123	Rao
CS52S	OS	345	Allen
CS52T	Networking	345	Allen
CS51S	ES	456	Mansingh

$\Pi_{Courseld,ename}$ Course ⋈ Instructor

Courseld	ename
CS51T	Rao
CS52S	Allen
CS51S	Mansingh

Inner Join

•In Inner join, tables are joined together where there is the match (=) of the primary and foreign keys.

R⋈<R.primary_key = S.foreign_key>S

Inner joins return rows only when there is at least one row from both tables that matches the join condition.

Inner joins eliminate the rows that do not match with a row from the other table

Student Table

Studid	name	course
100	Jayesh	PH
200	Preetesh	CM
300	Pramod	CM

Course Table

course#	name
PH	Pharmacy
CM	Computing

Students ⋈ course = course# Courses

Studid	name	course	course#	Course.name
100	Jayesh	PH	PH	Pharmacy
200	Preetesh	CM	CM	Computing
300	Pramod	CM	CM	Computing

Outer Join

•Inner join + rows of one table which do not satisfy the condition.
•Left Outer Join:
•R<R.primary_key = S.foreign_key>S
•All rows from R are retained and unmatched rows of S are padded with NULL
Student Table

Student Table

Studid	name	Course#
100	Jayesh	PH
200	Preetesh	CM
400	Pramod	EN

Course Table

course#	Cname
PH	Pharmacy
CM	Computing
CH	Chemisty

Пe.studid,e.name,e.Course#,c.Course#,c.Cname(ρ e (Student)=⋈ ρ c (Course)

e.studid	e . name	e.course#	c.course#	c.Cname
100	Jayesh	PH	PH	Pharmacy
200	Preetesh	CM	CM	Computing
400	Pramod	EN	NULL	NULL

Right Outer Join

•Right Outer Join:
R<R.primary_key = S.foreign_key>S
All rows from S are retained and unmatched rows of R are padded with NULL
Right outer Join takes all the record form the right relation S that unmatched any record in the S relation

Student Table

Studid	name	Course#
100	Jayesh	PH
200	Preetesh	CM
400	Pramod	EN

Course Table

course#	Cname
PH	Pharmacy
CM	Computing
CH	Chemisty

Πe.studid,e.name,e.Course#,c.Course#,c.Cname
(ρ e (Student)⋈= ρ c (Course)

e.studid	e . name	e.course#	c.course#	c.Cname
100	Jayesh	PH	PH	Pharmacy
200	Preetesh	CM	CM	Computing
NULL	NULL	NULL	CH	Chemisty

Full Outer Join

In Full outer join tables on the both sides of operator contains null values
It will contain record from both relations that do not join with any record from the other relation. Those tuples will be padded with NULLs as usual.

R COLA	R COLB
A	1
B	2
D	3
F	4
E	5

S COLA	S COLB
A	1
C	2
D	3
E	4

R.ColA = S.SColA

A	1	A	1
D	3	D	3
E	5	E	4
B	2	NULL	NULL
F	4	NULL	NULL
NULL	NULL	C	2

Relational Division Operator

•It is denoted as ÷.Let r(R) and s(S) be relations r÷ s:

-the result consists of the restrictions of tuples in r to the attribute names unique to R, i.e. in the Header of r but not in the Header of s, for which it holds that all their combinations with tuples in s are present in r.

Relation or table "r":

A	B
a	1
b	2
a	2
p	3
p	4

Relation or table "s":-

B
2
3

Thefore r%s

A
b
a
p

Aggregation Operators

Operators that summarize or aggregate the values in a single attribute of a relation.
Operators are the same in relational algebra and SQL.
All operators treat a relation as a bag of tuples.
SUM: computes the sum of a column with numerical values.
AVG: computes the average of a column with numerical values.
MIN and MAX:
for a column with numerical values, computes the smallest or largest value, respectively. for a column with string or character values, computes the lexicographically smallest or largest values, respectively.

COUNT: computes the number of non-NULL tuples in a column.
In SQL, can use COUNT (*) to count the number of tuples in a relation.
Grouping operator
γ L(R) where L is a list of items in the Relation(R) that are either
a) They ate individual attributes or grouping attributes or b)θ (A), Where θ is an aggregation operator and A the attribute in the relation(R) to which the aggregation operator is to applied It is computed by:
1. Group R according to all the grouping attributes on list L.
2. Within each group, compute θ(A), for each element θ(A) on list L.
3. Result is the relation whose columns consist of one tuple for each group. The components of that tuple are the values associated with each element of L for that group.

Let R =

Mall	Jeans	Price
R-Mall	Killer	1500
Metro Mall	Lee	1700
Phoenix Mall	Live's	1800
INORBIT *MALL*	Killer	1900
Spykar	Lee	1400

Compute γ Jeans,AVG(Price)

Group by the grouping attribute(s), Jeans in this case:

Mall	Jeans	Price
R-Mall	Killer	1500
INORBIT *MALL*	Killer	1900
Metro Mall	Lee	1700
Spykar	Lee	1400
Phoenix Mall	Live's	1800

Compute average of price within groups:

Jeans	Price
Killer	3400
Lee	3100
Live's	1800

Relational Calculus:

Relation calculus comes from one of the mathematical branches or logic is called predicate calculus.
The differentiate between the relational algebra and relation calculus : relational algebra provides a series of procedures that is used for solving the problem and relational algebra is describe what is problem is
It is closer than relational algebra to how users would formulate queries in terms of t
heir information needs, rather than in terms of operations.
Relational calculus is categories in to two part
1)Tuple relational calculus.
2)Domain relational Calculus.
Relational Calculus is an non procedural language whereas relational algebra is procedural Language.

TUPLE RELATIONAL CALCULUS

Tuple calculus is a calculus that was introduced by Edgar F.Codd as part of the relational model, in order to provide a declarative database-query language for this data model.
Tuple relational calculus is a non-procedural language
We must provide a formal description of the information desired.
Each queries in the Tuple Relation calculus is written as
{t| P(t) } i.e It is set of tuples t for which predicate P is true
We can also use the notation for describing the tuple calculus
We use t[a] to indicate the value of tuple t on attribute a
We use t∈r to indicate that tuple t is in relation r

Selection and Projection

To find out the data from the table we have to use the operator known as selection and projection that used to select desired data by applying some predicate calculus or formula on table
In Tuple relation Calculus a query can be written for selection operation(σ) as
σ p(r)={t| P(t) }
Where as σ p
(r)= Selection operator on the Relation
t= Set of tuples(as called as variable range over tuples)
p= Predicate indicate that is true for t
each formula in the relation calculus is consist of Connectivity by logical operator such as (∧), or (v), not (¬)
Set of comparison operators: (e.g.,<,≤,=,≠,>,≥)
Implication (⇒): x⇒y, if x if true, then y is true x⇒y≡¬xvy
Set of quantifiers:
□∃t∈r (Q(t))≡"there exists" a tuple in t in relation r
such that predicate Q(t) is true
□∀t∈r(Q(t))≡Q is true "for all" tuples t in relation r
Query to select all attributes of the table
Consider a sample database of an Employee

SSN	FirstName	LastName	Salary
101	Jayesh	Shinde	30000
102	Preetesh	Shinde	40000
103	Sachin	Tendulkar	50000
104	Pravin	Kanetkar	35000
105	Mahesh	Jadhav	53000

We Want to find all the record of employee table using relational calculus
Select all the Employee whose having the salary more than 30000
σ salary>30000(Employee)={t|t∈Employee∧t[Salary]>30000}

SSN	FirstName	LastName	Salary
102	Preetesh	Shinde	40000
103	Sachin	Tendulkar	50000
104	Pravin	Kanetkar	35000
105	Mahesh	Jadhav	53000

Query 2 Find the SSN for each Employee whose having salary more than 30000
ΠSSN(σ Salary>30000(Employee))={t|∃t∈Employee[SSN]=t[SSN]∧t[Salary]>300000}

SSN
102
103
104
105

SET Operations

In set operation, two or more select statement is combined together to form as desired result On other hand the set operation combines rows from two or more different queries
In select statement, there must be same number of columns retrieve from the two or more queries
There must be same data tape or compatible type of each columns in select statement
In tuple relational calculus ,a query can be written as r∪s={t|t∈r or t∈ s}
Where, t=Set of tuples(as called as variable range over tuples)
p= Predicate indicate that is true for t

Reserves Table

SID	BID	Day
22	101	10/10/96
95	103	11/12/96

Sailor Table

SID	Sname	rating	age
22	Jayesh	7	45.0
31	Preetesh	8	55.5
95	Pramod	3	63.5

Find the all Sailors ID whose rating is greater than 2 ,here we use the union operator in the relational algebra, In the relational calculus we used two exists clause and Connected by or
ΠSID(Reserves)∪ΠSID(Sailor)={t|∃s∈Reserves(t[SID]=s[SID])v∃u∈
Sailor (t[SID]=u[SID]∧rating>2}

SID
22
31
95

In This above Example duplicate record is eliminated Query to select the data using Intersection operator Find the those Sailors ID whose rating is greater than 7, here we use the Intersect operator in the relational algebra, In the relational calculus we used two exists clause and Connected by And ⊓ SID(Reserves)∩⊓ SID(Sailor)={t|∃s∈ Reserves(t[SID]=s[SID])∧∃u∈Sailor (t[SID]=u[SID]∧rating>7}

SID
31

In This above Example duplicate record is eliminated
Query to select data using difference operator

Cartesian product Operation

In Cartesian product operation defines as every tuples of relation R combines with every relation S
In the Relational Cartesian Product , The result will return as all the attributes from both relation R and S.
Syntax
r x s={t∈q| t∈r and r∈s}
Consider two table Employee and Department

EmployeeID	Designation
101	Lecturer
102	Assistant Professor
103	Professor

DepartNumber	DepartName
E1	Electrical
C1	Computer
E3	Electronics

In The tuple Relational calculus , requires two exits clause they are connected by ∧
The Query Can be Written as Π
EmployeeID(EmployeeXDepartment)={t|∃s∈Employee∧∃u∈Department}

EmployeeID	Designation	DepartNumber	DepartName
101	Lecturer	E1	Electrical
102	Assistant Professor	E1	Computer
103	Professor	E1	Electronics
101	Lecturer	C1	Electrical
102	Assistant Professor	C1	Computer
103	Professor	C1	Electronics
101	Lecturer	E3	Electrical
102	Assistant Professor	E3	Computer
103	Professor	E3	Electronics

Join Operator

Join operator is used to retrieves data from multiple relations

Syntax

r ⋈ s={t∈q|t∈r and r∈s}

Example

Retrieves data from the two table known as Employee and Department

EmployeeID	Designation	DepartNumber
101	Lecturer	E1
102	Assistant Professor	C1
103	Professor	E3

DepartNumber	DepartName
E1	Electrical
C1	Computer
E3	Electronics

In The tuple Relational calculus , requires two exits clause they are connected by∧

Find the Employee Id whose teaches in the Computer Department Π EmployeeID (Employee Department)={t|∃s∈Employee (t[EmployeeID]=s[EmployeeID]u∈ Department(u[DepartNumber]=s[DepartNumber]∧u[DepartName]='Computer'))}

EmployeeID
102

Division Operator

The division of relation R over relation S is denoted by R% S

Consider an example Student table has two attributes Student Name and Marks and another table is Marks

Student name	Marks
Dinesh	97
Arun	100
Kamal	98
Jay	85
Virat	98
Mahendra	95
Dharmendra	95

Marks
98

Domain Relational Calculus

In computer science, domain relational calculus(DRC) is a calculus that was introduced by Michel Lacroix and Alain Pirotte as a declarative database query language for the Relational data model.

In DRC,queries have the form: where each Xi is either a domain variable or constant, And denotes a DRC formula. The result of the query is the set of tuples X i to Xn which makes the DRC formula true.

This language uses the same operators as tuple calculus, the logical connectives ∧(and),∨(or) and ¬ (not). The existential quantifier(∃) and the universal quantifier(∀) can be used to bind the variables.

Its computational expressiveness is equivalent to that of Relational algebra

Example

Domain Relational Calculus

1)Find the names of all Clerks who earn more than RS 10,000.

{fN, lN | (sN, posn, sex, DOB, sal, bN)(Staff (sN, fN, lN, posn, sex,DOB, sal, bN) posn = 'Clerks'sal > 10,000)}

2) List the staff who manage properties for rent in Mumbai.{sN, fN, lN, posn, sex, DOB, sal, bN | (sN1,cty)

(Staff(sN,fN,lN,posn,sex,DOB,sal,bN) PropertyForRent(pN, st, cty,pc, typ, rms, rnt, oN, sN1, bN1) (sN=sN1) cty='Mumbai')

3)List the names of staff who currently do not manage any properties for rent.

{fN, lN| (sN) (Staff(sN,fN,lN,posn,sex,DOB,sal,bN) (~(sN1) (PropertyForRent(pN, st, cty, pc, typ, rms, rnt, oN, sN1, bN1)(sN=sN1))))}

4)Retrieve names of all professors who have taught Management345

{ N∃I∈ professor.Id ∃D ∈ Professor.Dept.Id (Professor(I,N,D)AND∃S∈ Teaching.Semester (Teaching(I,MGT345,S)))}

This can be abbreviated{N | Professor (I, N, D) AND(Teaching(I,MGT345,S)}

1)All courses that have been taken by every student:{ C| Course(D,C,C,D) AND∀S∈ Students.Id(Transcript(S,C,SEM,G))}

2)Find all students who have ever taken a course from every professor who has ever taught a course.

{I | Transcript(S,C,SEM,G1) AND ∀PI∈ Teaching .ProfId (Teaching(PI,C2,SEM2) AND Transcript(S,C,SEM,G2)

3)Retrieve IDs of students who did not take any courses in F2001:{I

|Student (I, N, A, S} AND NOT Transcript (I, C, F2001, G)}

4)Find potential student graders for this semester's courses:{P, C, S| Teaching (P, C, S2002) AND Transcript(S, C, SEM, G AND SEM<> S2002}
5)Find all loan numbers for loans with an amount greater than $1200:
{ < l > |∃ a, b (< l, a, b > ∈ loan∧ a > 1200) }
Equivalent Relational Algebra expression Π loan_number
(σamount > 1200(loan))
10)Find the loan numbers of all loans made jointly to Amit and Ramesh.
{ < l > |∃ x (<x, l> ∈ borrower∧x = "Amit")
∧ ∃ x (<x, l> ∈borrower∧x = "Ramesh")}
11)Find the names of all customers who have a loan from the Kurla branch, and find the loan amount.
{ < c,a > |∃ l (<c, l> ∈ borrower∧∃ b (<l, b, a> ∈ loan∧b = "Kurla")}
12)Find branch name, loan number, customer name and amount for loans of over $1200.
{<b,l,c,a>|< b,l,c,a >∈ borrow ∧ a> 1200
13)Find all customers who have a loan for an amount > than $1200.
{<c> |∃b ,l,a(<b,l,c,a>∈ borrow∧ a> 1200)}
14)Find all customers having a loan from the MTU branch, and the city in which they live.
{<c,x> |∃b ,l, a(<b,l,c,a>∈ borrow∧ b ="MTU" ∧ ∃y (<c,y,x >∈Customer))}
15)Find all customers having a loan, an account or both at the MTU branch.
{<c,x> |∃b ,l, a(<b,l,c,a>∈ borrow∧ b ="MTU")∨ ∃ b,a,n(<b,a,c,n>∈ deposit∧ = "MTU" }
16)Find all customers who have an account at all branches located in Kurla.
{<c > |∀ x,y,z (¬ (<x,y,z>∈ branch∧ z≠"Kurla") ∨ (∃,a,n(<x,a,c,n>∈ deposit)))}

RELATIONAL ALGEBRA VS RELATIONAL CALCULUS

Sr.No	Relational Algebra	Relational Calculus
1	Relational Algebra is a procedural query language, very useful for representing execution plans, relatively close to SQL.	The tuple Relational calculus is a non-procedural language, Lets users describe what they want, rather than how to compute it.
2	Relational algebra indicates operation on table that ptoduces a new tables ar a result	Relation Calculus defines a new table by providining representation in term of given relation
3	In relation algebra , A query can be written with help of relational operator known as selection, projection etc. $\Pi_{Columnname}(\sigma_{Condition}(<TableName>)$ Table is name of the input relation	In Relational Calculus A query Can be written as {t|P(t)} I.e The set of tuple t where Predicate P is true
4	In relation algebra , we need provide a series of procedures that use to generated the answer to respond of set of query	In relational Calculus we needs to provides a formal description of the information

CHAPTER VIII: CONSTRAINTS

INTRODUCTION

Definition:

Constraints are used to limit the type of data that can go into a table.
Constraints can be specified when a table is created (with the CREATE TABLE statement) or after the table is created (with the ALTER TABLE statement).

Syntax:

```
Create table table_name
{
Column data_type[column_constraint_Name][Column_constraint],
Column datatype[DEFAULT expr] [column_constraint],
.....................
[table_contraint][....]
}
```

Example:

Some attributes in the table are not required so such columns can be defined as NULL constraint. In the EMPLOYEE table it is allowed insert row having Phone number column as NULL.

```
Create table EMPLOYEE
{
Did varchar(10),
EName varchar(10),
Phone_Number char (100) NULL
}
```

Data Integrity

Constraints are used to enforce the data integrity. This ensures the accuracy and reliability of the data in the database.
The following categories of the data integrity exist:
¬Entity Integrity
¬Domain Integrity
¬Referential integrity
¬User-Defined Integrity
Entity Integrity ensures that there are no duplicate rows in a table.
Ex: Unique, Primary Key

Domain Integrity

enforces valid entries for a given column by restricting the type, the format, or the range of possible values.
Ex: check, Null, not Null

Referential integrity ensures that rows cannot be deleted, which are used by other records (for example, corresponding data values between tables will be vital).
Ex: Foreign Key

User-Defined Integrity

enforces some specific business rules that do not fall into entity, domain, or referential integrity categories.
TYPES OF CONSTRAINTS
Constraints can be defined in two ways:-
1) The constraints can be specified immediately after the column definition. This is called column-level definition.
2) The constraints can be specified after all the columns are defined. This is called table-level definition.

Some of the Constraints are listed below:
¬NOT NULL
¬UNIQUE
¬PRIMARY KEY
¬FOREIGN KEY
¬CHECK

NOT NULL CONSTRAINT

•The NOT NULL constraint enforces a column to NOT accept NULL values.
•The NOT NULL constraint enforces a field to always contain a value. This means that you cannot insert a new record, or update a record without adding a value to this field.
Syntax to define a Not Null constraint:
[CONSTRAINT constraint name] NOT NULL
For Example:
To create an employee table that enforces the "E_Id" column and the "LastName" column to not accept NULL values:

```
Create Table EMPLOYEE
(
E_Id int NOT NULL,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Addre
ss varchar(255),
City varchar(255)
)
```

¬**UNIQUE KEY CONSTRAINT**

This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.
Syntax to define a Unique key at column level:
[CONSTRAINT constraint_name] UNIQUE
Syntax to define a Unique key at table level:
[CONSTRAINT constraint_name] UNIQUE(column_name)
For Example:
To create an employee table with Unique key, the query would be like,

Unique Key at column level:

```
CREATE TABLE employee
(E_Id number(5) PRIMARY
KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) UNIQUE
);
```

OR

```
CREATE TABLE employee
(E_Id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10) CONSTRAINT
loc_un UNIQUE
);
```

Unique Key at table level:

```
CREATE TABLE employee
(E_Id number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
salary number(10),
location char(10),
CONSTRAINT loc_un UNIQUE(location)
);
```

¬PRIMARY KEY CONSTRAINTS:

The PRIMARY KEY constraint uniquely identifies each record in a database table.
Primary keys must contain unique values.
A primary key column cannot contain NULL values.
Each table should have a primary key, and each table can have only ONE primary key.
Syntax to define a Primary key at column level:
column name datatype [CONSTRAINT constraint_name] PRIMARY KEY
Syntax to define a Primary key at table level:
[CONSTRAINT constraint_name] PRIMARY KEY(column_name1,column_name2,..)
•column_name1, column_name2 are the names of the columns which define the primary Key.
•The syntax within the bracket i.e. [CONSTRAINT constraint_name] is optional.
For Example:
To create an employee table with Primary Key constraint, the query would be like.

Primary Key at table level:

CREATE TABLE employee (E_Id number(5) PRIMARY KEY, name char(20), dept char(10), age number(2), salary number(10), location char(10)); **OR**	CREATE TABLE employee (E_Id number(5) CONSTRAINT emp_id_pk PRIMARY KEY, name char(20), dept char(10), age number(2), salary number(10), location char(10));

Primary Key at table level:

```
CREATE TABLE employee(E_Id number(5),name char(20),dept char(10),age
number(2),salary number(10),location char(10),CONSTRAINT emp_id_pk PRIMARY KEY (id)
);
```

¬FOREIGN KEY CONSTRAINT

•This constraint identifies any column referencing the PRIMARY KEY in another table.
•It establishes a relationship between two columns in the same table or between different tables.
•For a column to be defined as a Foreign Key, it should be a defined as a Primary Key in the table which it is referring.

One or more columns can be defined as foreign key.
Syntax to define a Foreign key at column level:
[CONSTRAINT constraint_name] REFERENCES Referenced_Table_name(column_name)
Syntax to define a Foreign key at table level:
[CONSTRAINT constraint_name] FOREIGN KEY(column_name)
REFERENCES referenced_table_name(column_name);
For Example:
1) Lets use the "product" table and "order_items".
Foreign Key at column level:

Foreign Key at column level:

```
CREATE TABLE product
( product_id number(5)
CONSTRAINT pd_id_pk
PRIMARY KEY,
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
```

OR

```
CREATE TABLE order_items
( order_id number(5)
CONSTRAINT od_id_pk
PRIMARY KEY,
product_id number(5)
CONSTRAINT pd_id_fk
REFERENCES,
product(product_id),
product_name char(20),
supplier_name char(20),
unit_price number(10)
);
```

Foreign Key at table level:

```
CREATE TABLE order_items
( order_id number(5) ,
product_id number(5),
product_name char(20),
supplier_name char(20),
unit_price number(10)
CONSTRAINT od_id_pk PRIMARY KEY(order_id),
CONSTRAINT pd_id_fk FOREIGN KEY(product_id)
REFERENCES product(product_id));
```

2) If the employee table has a 'mgr_id' i.e, manager id as a foreign key which references primary key 'id' within the same table, the query would be like,

```
CREATE TABLE employee
(E_Id
number(5) PRIMARY KEY,
name char(20),
dept char(10),
age number(2),
mgr_id number(5) REFERENCES employee(id),
salary number(10),
location char(10)
);
```

¬CHECK CONSTRAINTS:

This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.
Syntax to define a Check constraint:
[CONSTRAINT constraint_name] CHECK (condition)
For
Example:
In the employee table to select the gender of a person, the query would be like
Check Constraint at column level:

```
CREATE TABLE employee
(E_Id number(5) PRIMARY KEY,name char(20),dept char(10),age number(2),
gender char(1) CHECK (gender in('M','F')),salary number(10),location char(10));
```

Check Constraint at table level:

```
CREATE TABLE employee
(E_Id number(5) PRIMARY KEY,name char(20),dept char(10),age number(2),gender
char(1),salary number(10),location char(10),
CONSTRAINT gender_ck CHECK (gender in ('M','F')));
```

INTEGRITY CONSTRAINTS

- Constraints are used to maintain integrity of database so they are also called as data Integrity constraints
- Data integrity constraints provide a way of ensuring that changes made to the database by authorized users do not result in a loss of data consistency and correctness.
- An integrity constraint can be any arbitrary predicate or condition applied to the database.
- Integrity constraints may be difficult to evaluate, so will only consider integrity constraints that can be tested easily with minimal overhead.
- Integrity constraint with E-R models
- Key declarations: ability that the certain attributes of relations can form a candidate key for a given entity set.
- Form of a relationship
- : Mapping cardinalities like 1:1, 1-Many and Many to many.
- To maintain integrity in the database we have many types of constraints which can keep database in integrity state.

CHAPTER IX: VIEWS

Definition:

- A view is a virtual table that consists of columns from one or more tables.
- A virtual table is like a table containing fields but it does not contain any data. In run time it contains the data and after that it gets free.
- But table stores the data in database occupy some space.
- Just like table, view contains Rows and Columns which is fully virtual based table.
- Base Table-The table on which view is defined is called as Base table.

CREATING A VIEW

This statement is used to create a view.

Syntax:

CREATE VIEW view_name

- The CREATE statement assigns a name to the view and also gives the query which defines the view.
- To create the view one should must have privileges to access all of the base tables on which view is defined.
- The create view can change name of column in view as per requirements.

HORIZONTAL VIEW

A Horizontal view will restrict the user's access to only a few rows of the table.

Example:

Define a view for Sue (employee number 1004) containing only orders placed by customers assigned to her.

```
CREATE VIEW SUEORDERS AS SELECT * FROM ORDERS
WHERE CUST IN(SELECT CUST_NUM FROM CUSTOMERS WHERE CUST_REP=1004)
```

VERTICAL VIEW

A vertical view restricts a user's access to only certain columns of a table.

Ex:

```
CREATE VIEW EMP_ADDRESS AS
SELECT EMP NO, NAME, ADDR1, ADDR2, CITY
FROM EMPLOYEE ROW/COLUMN SUBSET VIEW.
```

- Views can be used to restrict a user to access only selected set of rows and columns of a table in a database.
- This view generally helps us to visualize how view can represent the base table.
- This type of view is combination of both horizontal and vertical views.

Ex:

```
CREATE VIEW STUDENTS_PASSED AS
SELECT ROLLNO, NAME, PERCENTAGE
FROM STUDENTS
WHERE RESULT ='PASS'
```

GROUPED VIEW

- A grouped view is one in which query includes GROUPBY CLAUSE.
- It is used to group related rows of data and produce only one result row for each group.

Ex:

Find summary information of Employee Salaries in sales Department.

Defining View

```
CREATE VIEW Summary_Empl_Sal
(
Total_Employees,
Minimum_salary,
Maximum_Salary,
Average_salary,
Total_salary
```

)
AS
SELECT COUNT(EmpID),
Min(Salary),
Max(Salary),
Avg(Salary),
SUM(Salary),
FROM Employee
GROUP BY Department
HAVING Department='Sales';
View Call
Select *From Summary_Empl_Sal
The above Query will give,
Total No. Of Employees in sales Department, Minimum Salary in sales Department. 4Maximum Salary in sales Department. Average Salary in sales Department. Total Salary of Employees in sales Department.

JOINED VIEWS

- A Query based on more than one base table is called as Joined View.
- It is also called as Complex View
- This gives a way to simplify multi table queries by joining two or more table query in the view definition that draws its data from multiple tables and presents the query results as a single view.
- The view once it is ready we can retrieve data from multiple tables without joining any table simply by accessing a view created.

Ex:
Company database find out all EMPLOYEES for respective
DEPARTMENTS.Schema
Definition:
EMPLOYEE-> EmpID, EmpName, Salary, DeptID
DEPARTMENT-> DeptID,DeptName
View
Definition
CREATE VIEW Emp_Details As
Select Employee,EmpID,
Department, DeptID,
Department, DeptName From
Where Employee.DeptID=Department.DeptID;
View Call
Select * from Emp_Details

DROPPING VIEW

When a view is no longer needed, it can be removed by using DROP VIEW statement.
Syntax:
DROP VIEW
<VIEW NAME> [CASCADE/RESTRICT]
CASCADE: It deletes the view with all dependent view on original view.
RESTRICT: It deletes the view only if they're in no other view depends on this view.
Example:
Consider that we have view VABC and VPQR .View VPQR depends on VABC.
Query:
DROP view VABC
If we drop VABC, then cascading affect takes place and view VPQR is also dropped.
Thus default option for dropping a view is CASCADE. The CASCADE option tells DBMS to delete not only the named view,but also query views that depend on its definition.
But,QUERY:
DROP view VABC RESTRICT

Here, the query will fail because of RESTRICT option tells DBMS to remove the view only if no other views depend on it. Since VPQR depends on VABC, will cause an error.

UPDATING VIEWS

•Records can be updated, inserted, and deleted though views.
•UPDATAEBLE VIEWS are those in which views are used against INSERT, DELETE and UPDATE statements.
The following conditions must be fulfilled for view updates:
•DISTINCT must not be specified; that is, duplicate rows must not be eliminated from the query results.
•The FROM clause must specify only one updateable table; that is, the view must have a Single source table for which the user has the required privileges. If the source table is itself a view, then that view must meet these criteria.
•Each select item must be a simple column reference; the select list cannot contain expressions, calculated columns, or column functions.
•The WHERE clause must not include a sub query; only simple row-by-row search conditions may appear.
•The query must not include a GROUP BY or a HAVING clause.

ADVANTAGES OF VIEWS

1.**Security**
Each user can be given permission to access the database only through a small set of views that contain the specific data the user is authorized to see, thus restricting the user's access to stored data.
2.**Query simplicity**
A view can draw data from several different tables and present it as a single table, turning multi table queries into single-table queries against the view.
3.**Structural simplicity**
Views can give a user a personalized view of the database structure, presenting the database as a set of virtual tables that make sense for that user.
4.**Insulation from change**
A view can present a consistent, unchanged image of the structure of the database, even if the underlying source tables are split, restructured ,or renamed. Note, however, that the view definition must be updated whenever underlying tables or columns referenced by the view are renamed.
5.**Data integrity**
If data is accessed and entered through a view, the DBMS can automatically check the data to ensure that it meets specified integrity constraints.

DISADVANTAGES OF VIEWS

While views provide substantial advantages, there are also three major disadvantages to using a view instead of a real table:
•Performance
Views create the appearance of a table, but the DBMS must still translate queries against the view into queries against the underlying source tables.
If the view is defined by a complex multi table query, then even a simple query against the view becomes a complicated join, and it may take a long time to complete.
However, the issue isn't because the query is in a view—any poorly constructed query can present performance problems—the hazard is that the complexity is hidden in the view, and thus users are not aware of how much work the query is performing.
•Manageability
Like all database objects, views must be managed. If developers and database users are allowed to freely create views without controls or standards, the DBA's job becomes that much more difficult.
This is especially true when views are created that reference other views, which in turn reference even more views.
The more layers between the base tables and the views,
the more difficult it is to resolve problems attributed to the views.

•Update restrictions

When a user tries to update rows of a view, the DBMS must translate the request into an update on rows of the underlying source tables.

This is possible for simple views, but more complex views cannot be updated; they are read-only.

COMPARISON BETWEEN TABLES AND VIEWS

VIEWS

•View comprises of Query in view definition.

•Just like table, view contains Rows and columns which is fully virtual based table.

•The fields in a view are fields from one or more real tables in the database.

•When view is called, it does not contain any data. For that, it goes to memory and fetches data from base table and displays it.

•E-g:-

An I.T. Faculty requires only I.T. related data of students so we can create view called as Stud_IT_View for Faculty as below which will only depicts I.T. data of students to I.T. faculty.

•A virtual table is like a table containing fields but it does not contain any data. In run time it contains the data and after that it gets free. But table stores the data in database occupy some space.

Stud_IT_View (Student_Id,Student_Name, I.T.)

We can also add functions like WHERE and JOIN statements to a view and present the data as if the data were coming from one single table.

TABLES

•Table stores the data and database occupies some space in database.

•Tables contain rows and columns, columns representing fields and rows containing data or records.

EX:Consider a Employee containing following columns,

EMPLOYEE (Emp_ID, EmpName, Designation, Address, Salary)

CHAPTER X: STRUCTURE QUERY LANGUAGE(SQL)

SQL stands for Structured Query Language

- It lets you access and manipulate databases.
- SQL was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce in the early 1970s.
- The first commercial relational database was released by Relational Software (Later called as Oracle).
- SQL is not a case sensitive as it is a keyword based language and each statement begins with a unique keyword.

FEATURES OF SQL

- SQL can execute queries against a database.
- SQL can retrieve data from a database.
- SQL can insert ,Update, Delete, records in a database
- SQL can create stored procedures in a database
- SQL can create views in a database.

SQL COMMANDS:

- SQL commands are instructions used to communicate with the database to perform specific task that work with data.
- SQL commands can be used not only for searching the database but also to perform various other functions like, for example, you can create tables, add data to tables, or modify data, drop the table, set permissions for users.
- SQL commands are grouped into 2 major categories depending on their functionality:

Data Definition Language(DDL)-

These SQL commands are used for creating, modifying, and dropping the structure of Database objects. The commands are CREATE, ALTER, DROP,RENAME, and TRUNCATE.

Data Manipulation Language (DML)-

These SQL commands are used for storing, retrieving, modifying, and deleting data. These commands are SELECT, INSERT, UPDATE, and DELETE.

DATA DEFINITION LANGUAGE(DDL)•

DDL statements are used to build and modify the objects and structure of tables in database.

- The DDL part of SQL permits database tables to be created or deleted.
- It also defines indexes (keys), specifies links between tables, and imposes constraints between tables.
- The most important DDL statements in SQL are:
- CREATE TABLE-creates a new table
- ALTER TABLE-modifies a table
- DROP TABLE-deletes a table
- CREATE INDEX-creates an index (search key)
- DROP INDEX-deletes an index

a.

CREATE COMMAND

This statement used to create Database.

Syntax:

```
CREATE TABLE tablename
(
column_name data_type attributes...,
column_name data_type attributes...,
...
)
```

- Table and column names can't have spaces or be "reserved words" like TABLE, CREATE, etc.

3
Example:
CREATE TABLE Employee
(
EmpId varchar2(10),
FirstName char(20),
LastName char(20),
Designation char(20),
City char(20)
);
OUTPUT

Emp_Id	FirstName	LastName	Designation	City

ALTER COMMAND:

•This statement is used to make modifications to the table structure.
•This statement is also used to add, delete, or modify columns in an existing table
Syntax:
ALTER TABLE table_name
ADD column_name datatype
OR
ALTER TABLE table_name
DROP COLUMN column_name
OR
ALTER TABLE table_name
MODIFY COLUMN column_name

Example:

ALTER
TABLE Employee ADD DateOfBirth date

EMP_Id	FirstName	LastName	Designation	City	DateOfBirth
1	Raj	Malhotra	Manager	Mumbai	
2	Henna	Setpal	Executive	Delhi	

DROP COMMAND:

This statement is used to delete a table.
Emp_Id
FirstName
LastName
Designation
City

Syntax:

DROP TABLE table_name

Example:

DROP TABLE Employee

DATA MANIPULATION LANGUAGE(DML)

DML is set of commands used to,

- Insert data into table
- Delete data from table
- Update data of table.

EMP_Id	FirstName	LastName	Designation	City
1	Raj	Malhotra	Manager	Mumbai
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore

a.**INSERT**

The INSERT statement is used to insert a new row in a table.

Syntax:

INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

Example:

INSERT INTO Employee VALUES (4,'Nihar)
INSERT INTO Employee VALUES (5,'savita)
INSERT INTO Employee VALUES (6,'Diana)

OUTPUT:

Emp_Id	FirstName
4	Nihar
5	Savita
6	Diana

b.**DELETE**

The DELETE statement is used to delete records in a table.

Syntax:

DELETE FROM table_name
WHERE some_column=some_value

Example:

EMP_Id	FirstName	LastName	Designation	City
1	Raj	Malhotra	Manager	Mumbai
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore

DELETE FROM Employee WHERE LastName='Malhotra' AND FirstName='Raj'

EMP_Id	FirstName	LastName	Designation	City
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore

c.UPDATE

The UPDATE statement is used to update records in a table.

Syntax:

UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value

Example:

EMP_Id	FirstName	LastName	Designation	City
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore
4	Nihar	Sarkar		

```
UPDATE Employee
SET Designation='CEO, City='Mumbai'
WHERE LastName='Sarkar' AND FirstName='Nihar'
```

OUTPUT:

EMP_Id	FirstName	LastName	Designation	City
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore
4	Nihar	Sarkar	CEO	Mumbai

SQL BASIC QUERIES

a. SELECT CLAUSE

This statement is used for various attributes or columns of a table.
SELECT can have 2 options as SELECT ALL OR SELECT DISTINCT, where SELECT ALL is default select all rows from table and SELECT DISTINCT searches for distinct rows of outputs.
Syntax:

```
SELECT
*
FROM
table_name
```

b.**FROM CLAUSE**

This clause is used to select a Relation/Table Name in a database.

c.**WHERE CLAUSE**

This clause is used to put a condition on a query result.
Example:
Ex1:
SELECT * FROM Employee

EmpID	FirstName	LastName	Designation	City
1	Raj	Malhotra	Manager	Mumbai
2	Henna	Setpal	Executive	Delhi
3	Aishwarya	Rai	Trainee	Indore

Ex 2:
select only the distinct values from the column named "City" from the table above.

```
SELECT DISTINCT City
FROM Employee
```

WHERE City='Mumbai'

Output:

EmpID	FirstName	LastName	Designation	City
1	Raj	Malhotra	Manager	Mumbai

Aliases

•SQL Aliases are defined for columns and tables.
•Basically aliases are created to make the column selected more readable.
Example:
To select the first name of all the students, the query would be like:
Aliases for columns:
SELECT
First Name AS Name FROM Employee;
or
SELECT First Name Name FROM Employee;
In the above query, the column
First Name is given a alias as 'name'.
So when the result is displayed the column name appears as
'Name' instead of 'FirstName'.

Output:

Name
Raj
Henna
Aishwarya
Nihar

Aliases for tables:

SELECT e.FirstName FROM Employee e;
In the above query, alias 'e' is defined for the table Employee and the column
First Name is selected from the table.
•Aliases is more useful when
•There are more than one tables involved in a query,
•Functions are used in the query,
•The column names are big or not readable,
•More than one columns are combined together

SQL ORDER BY

The ORDER BY clause is used in a SELECT statement to sort results either in ascending or descending order. Oracle sorts query results in ascending order by default.

Syntax

SELECT column-list FROM table_name [WHERE condition]
[ORDER BY column1 [, column2, .. columnN] [DESC]];

Database table "Employee";

EmpID	Name	LastName	Designation	Salary	City
1	Raj	Malhotra	Manager	56000	Mumbai
2	Henna	Setpal	Executive	25000	Delhi
3	Aishwarya	Rai	Trainee	20000	Indore

Example:
To select the entire
Employee from the table above, however, we want to sort the employee by their last name.
SELECT * FROM Employee
ORDER BY LastName

Database table "Employee";

EmpID	Name	LastName	Designation	Salary
1	Raj	Malhotra	Manager	56000
2	Henna	Setpal	Executive	25000
3	Aishwarya	Rai	Trainee	20000

ORDER BY DESC Clause
Using ORDER BY clause of a SELECT statement.
Example:
To select all the Employee from the table above, however, we want to sort the Employee descending by their last name.
SELECT * FROM Employee
ORDER BY LastName DESC
OUTPUT:

EmpID	Name	LastName	Designation	Salary	City
3	Aishwarya	Rai	Trainee	20000	Indore
2	Henna	Setpal	Executive	25000	Delhi
1	Raj	Malhotra	Manager	56000	Mumbai

AGGREGATE FUNCTIONS

SQL aggregate functions return a single value, calculated from values in a column.
Aggregate functions in SQL are as follows:
•AVG()–This functions returns the average value.
•COUNT()-This functions returns the number of rows.
•MAX()-This functions returns the largest value.
•MIN()-This functions returns the smallest value.
•SUM()-This functions returns the sum.

StudID	Name	Marks
1	Rahul	90
2	Savita	90
3	Diana	80
4	Heena	99
5	Jyotika	89
6	Rubi	88

AVG() Function

The AVG() function returns the average value of a numeric column.
This function first calculates sum of column and then divide by total number of rows.

Syntax:

SELECT AVG(column_name) FROM table_name

Example:

Find average Marks of Students from above table.
SELECT AVG(Marks) AS AvgMarks FROM Employees
The result-set will look like this:

AvgMarks
89.3

COUNT() Function

The COUNT() function returns the number of rows that matches a specified criteria.
Syntax:
SELECT COUNT(column_name) FROM table_name
Example
SELECT COUNT(StudID) AS Count FROM Students

Count
6

SUM() Function

The SUM() function returns the total sum of a numeric column.

Syntax

SELECT SUM(column_name) FROM table_name

Example

Find total of marks scored by students.
Select SUM (Marks) as Sum from Students

Output:

SUM
536

MIN() Function

The MIN() function returns the smallest value of the selected column.

Syntax
SELECT MIN(column_name) FROM table_name
Example
Find minimum scored by students
Select MIN(Marks) as Min from Students

Min
80

MAX() Function
The MAX() function returns the largest value of the selected column.
Syntax
SELECT MAX(column_name) FROM table_name
Example
Find maximum scored by students
Select MAX(Marks) as Max from Students
Max
90
NESTED SUB-QUERIES
•A query within a query is called Sub-Query.
•Subquery or Inner query or Nested query is a query in a query.
•Sub query in WHERE Clause (<>, ̂, =, <>): It is used to select some rows from main query.
•Sub query in HAVING Clause (IN/ANY/ALL)
: It is used to select some groups from main querySubqueries can be used with the following sql statements along with the comparison operators like =, <, >, >=, <= etc.
SYNTAX:
SELECT
select_Item
FROM
table_name
WHERE
expr_Operator
(SELECT select_item FROM Table_name)
Expression operator can be of 2 types:
1.Single Row Operator
2.Multiple-row Operator
Single Row Operator
A single-row subquery is one where the subquery returns only one value. In such a subquery you must use a single-row operator such as:

Operator	Description
=	Equal To
<>	Not Equal To
>	Greater Than
>=	Greater Than Equal To
<	Less Than
<=	Less Than Equal To

The single-row operators are used to write single-row Subqueries. The table below demonstrates the use of the single-row operators in writing single-row sub queries.

Operator	Query	Example
=	Retreive the details of employees who get the same salary as the employee whose ID is 101.	SELECT * FROM EMPLOYEES WHERE SALARY=(SELECT SALARY FROM EMPLOYEES WHERE EMPLOYEE_ID=101);
<>	Retreive the details of departments that are not located in the same location ID as department 10.	SELECT * ROM DEPARTMENTS WHERE LOCATION_ID <>(SELECT LOCATION_ID FROM DEPARTMENTS WHERE EPARTMENT_ID=10);
>	Retrieve the details of employees whose salary is greater than the minimum salary.	SELECT *FROM EMPLOYEES WHERE SALARY > (SELECT MIN(SALARY) FROM EMPLOYEES);
>	employees whose salary is greater than the minimum salary.	WHERE SALARY > (SELECT MIN(SALARY) FROM EMPLOYEES);
>=	Retrieve the details of employees who were hired on or after the same date that employee 201 was hired.	SELECT * FROM EMPLOYEES WHERE HIRE_DATE >= (SELECT HIRE_DATE FROM EMPLOYEES WHERE EMPLOYEE_ID=201);
<	Retrieve the details of employees whose salary is less than the maximum salary of employees in department 20.	SELECT * FROM EMPLOYEES WHERE SALARY < (SELECT MAX(SALARY) FROM EMPLOYEES WHERE DEPARTMENT_ID=20);
<=	Retrieve the details of employees who were hired on or before the same date that employee 201 were hired.	SELECT * FROM EMPLOYEES WHERE HIRE_DATE <=(SELECT HIRE_DATE FROM EMPLOYEES WHERE EMPLOYEE_ID=201);

A multiple row sub query is one where the sub query may return more than one value. In such type of subquery, it is necessary to use a multiple-row operator
The table below describes the multiple-row operators that can be used when writing multiple-row subqueries:

Operator	Meaning
IN	Equal to any value returned by the subquery
ANY	Compare value to each value returned by the subquery
ALL	Compare value to every value returned by the subquery

The multiple-row operators are used to write multiple-row subqueries. The table below demonstrates the use of the multiple-row operators in writing multiple-row subqueries.

Operator	Query	Example
IN	Retreive the department ID, department name and location ID of departments that are located in the same location ID as a location in the UK.	SELECT DEPARTMENT_ID, DEPARTMENT_NAME, LOCATION_ID FROM DEPARTMENTS WHERE LOCATION_ID IN (SELECT LOCATION_ID FROM LOCATIONS WHERE COUNTRY_ID='UK')
>ALL (Greater than the maximum returned by the subquery)	Retrieve the first name of employees whose salary is greater than the all the salaries of employees belonging to department 20.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEES WHERE DEPARTMENT_ID=20)
>ALL (Greater than the maximum returned by the subquery)	Retrieve the first name of employees whose salary is greater than the all the salaries of employees belonging to department 20.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEES WHERE DEPARTMENT_ID=20)
<ALL (Less than the least value returned by the subquery)	Retrieve the first name of employees whose salary is less than all the salaries of employees belonging to department 20.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY < ALL (SELECT SALARY FROM EMPLOYEES WHERE DEPARTMENT_ID=20)
>ANY (Greater than the minimum value returned by the subquery)	Retrieve the first name of employees whose salary is greater than the minimum salary of employees in department 60.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY > ANY (SELECT SALARY FROM EMPLOYEES WHERE DEPARTMENT_ID=60)

<ANY (Less than the maximum value returned by the subquery)	Retrieve the first name of employees whose salary is less than the maximum salary of employees in department 60.	SELECT FIRST_NAME FROM EMPLOYEES WHERE SALARY < ANY (SELECT SALARY FROM EMPLOYEES WHERE DEPARTMENT_ID=10)

EXISTS CLAUSE

•Exist Clause specifies a sub query to test for the existence of rows.
•Their results type is in BOOLEAN format.
•It Returns TRUE if a sub query contains any rows

Example:

SELECT * FROM suppliers WHERE EXISTS(select * from orders where suppliers.supplier_id = orders.supplier_id);

This select statement will return all records from the suppliers table where there is at least one record in the orders table with the same supplier_id.

NOT EXISTS CLAUSE

•The EXISTS condition can also be combined with the NOT operator.

Example:

SELECT * FROM suppliers WHERE not exists (select * from orders
Where suppliers.supplier_id = orders.supplier_id);

This will return all records from the suppliers table where there are no records in the Orders table for the given supplier_id.

NULL VALUES

•NULL values represent missing unknown data.
•By default, a table column can hold NULL values.
•If a column in a table is optional, we can insert a new record or update an existing record without adding a value to this column. This means that the field will be saved with a NULL value.
•NULL values are treated differently from other values.
•NULL is used as a placeholder for unknown or inapplicable values.

"Employee" table:

EmpId	FirstName	LastName	Address	City
1	Hussain	Lakdhwala		Santacruz
2	Elie	Sen	Juhu Road	Santacruz
3	Ranbir	Kapoor		Bhayander

Suppose that the "Address" column in the "Employee" table is optional.
This means that if we insert a record with no value for the "Address" column, the "Address" column will be saved with a NULL value.

IS NULL VALUES

How do we select only the records with NULL values in the "Address" column?
We will have to use the IS NULL operator:
SELECT FirstName, LastName, Address FROM Employee WHERE Address IS NULL

Output:

FirstName	LastName	Address
Hussain	Lakdhwala	
Ranbir	Kapoor	

IS NOT NULL VALUES

How do we select only the records with no NULL values in the "Address" column?
We will have to use the
IS NOT NULL operator:

```
SELECT LastName,FirstName, Address FROM Employee
WHERE Address IS NOT NULL
```

Output:

FirstName	LastName	Address
Elie	Sen	Juhu Road

JOINS

- Joins are used to relate information in different tables.
- A Join condition is a part of the sql query that retrieves rows from two or more tables.
- A SQL Join condition is used in the SQL WHERE Clause of select, update, delete statements.

Syntax for joining two tables is:

```
SELECT col1, col2, col3...
FROM table_name1, table_name2
WHERE table_name1.col2 = table_name2.col1;
```

If a sql join condition is omitted or if it is invalid the join operation will result in a Cartesian product. The Cartesian product returns a number of rows equal to the product of all rows in all the tables being joined.
Example:
If the first table has 20 rows and the second table has 10 rows, the result will be 20 * 10, or 200 rows.
This query takes a long time to execute.
Let us use the below two tables to explain the sql join conditions.

Database table "product";

Product_id	Product_name	Supplier_name	Unit_price
100	Camera	Nikon	300
101	Television	LG	100
102	Refrigerator	Videocon	150
103	IPod	Apple	75
104	Mobile	Nokia	50

Database table "order_items";

order_id	product_id	total_units	customer
5100	104	30	Infosys
5101	102	5	Satyam
5102	103	25	Wipro
5103	101	10	TCS

Joins can be classified into Equi join and Non Equi join.
1)SQL Equi joins
2) SQL Non equi joins
1)SQL Equi joins
•It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of equi joins are SQL Outer join and SQL Inner join.
Example:
We can get Information about a customer who purchased a product and the quantity of product. An equi-join is classified into two categories:
a) SQL Inner Join
b) SQL Outer Join
a) SQL Inner Join:
All the rows returned by the sql query satisfy the sql join condition specified.
Example:
To display the product information for each order the query will be as given below.
Since retrieving the data from two tables, you need to identify the common column between these two tables, which is the product_id.
QUERY:
SELECT order_id, product_name, unit_price, supplier_name, total_units
FROM product, order_items
WHERE order_items.product_id = product.product_id;
The columns must be referenced by the table name in the join condition, because product_id is a column in both the tables and needs a way to be identified.
b) SQL Outer Join:
•Outer join condition returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables.
•The syntax differs for different RDBMS implementation.
•Few of them represent the join conditions as ”LEFT OUTER JOIN" and "RIGHT OUTER JOIN".
Example
Display all the product data along with order items data, with null values displayed for order items if a product has no order item.
QUERY
SELECT p.product_id, p.product_name, o.order_id,o.total_units
FROM order_items o, product p WHERE o.product_id (+) = p.product_id;
Output:

Output:

Product_id	product_name	order_id	total_units
100	Camera		
101	Television	5103	10
102	Refrigerator	5101	5
103	IPod	5102	25

SQL Self Join:

A Self Join is a type of sql join which is used to join a table to it, particularly when the table has a FOREIGN KEY that references its own PRIMARY KEY.

It is necessary to ensure that the join statement defines an alias for both copies of the table to avoid column ambiguity.

Example

```
SELECT a.sales_person_id, a.name, a.manager_id,b.sales_person_id, b.name
FROM sales_person a, sales_person b
WHERE a.manager_id = b.sales_person_id;
```

2) SQL Non Equi Join:

A Non Equi Join is a SQL Join whose condition is established using all comparison operators except the equal (=) operator. Like >=, <=, <, >

Example:

Find the names of students who are not studying either Economics, the sql query would be like, (lets use Employee table defined earlier.)

QUERY:

```
SELECT first_name, last_name, subject FROM Employee WHERE subject != 'Economics'
```

Output:

Output:

first_name	last_name	subject
Anajali	Bhagwat	Maths
Shekar	Gowda	Maths
Rahul	Sharma	Science
Stephen	Fleming	Science

CHAPTER XI: TRIGGERS

TRIGGERS

A trigger is an operation that is executed when some kind of event occurs to the database. It can be a data or object change.

Creation of Triggers

- Triggers are created with the CREATE TRIGGER statement.
- This statement specifies that the on which table trigger is defined and on which events trigger will be invoked.
- To drop Trigger one can use DROP TRIGGER statement.

Syntax:

CREATE TRIGGER [owner.]trigger_name ON[owner.] table_name
FOR[INSERT/UPDATE/DELETE] AS IF UPDATE(column_name)
[{AND/OR} UPDATE(COLUMN_NAME)...] { sql_statements }

Triggers Types:

a.Row level Triggers
b.Statement Level Triggers

a.Row Level triggers-A row level trigger is fired each time the table is affected by the triggering statement.

Example:

- If an UPDATE statement updates multiple rows of a table, a row trigger s fired once for each row affected by the update statement.
- A row trigger will not run, if a triggering statement affects no rows.
- If FOR EACH ROW clause is written that means trigger is row level trigger.

b. Statement Level Triggers

A statement level trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement affects, even If no rows are affected.

Example:

- If a DELETE statement deletes several rows from a table, a statement level DELETE trigger is fired only once.
- Default when FOR EACH ROW clause is not written in trigger that means trigger is statement level trigger

Rules of Triggers

- Triggers cannot create or modify Database objects using triggers

O For example, cannot perform "CREATE TABLE... or ALTER TABLE" sql statements under the triggers

- It cannot perform any administrative tasks

O For example, cannot perform "BACKUP DATABASE..." task under the triggers

- It cannot pass any kind of parameters
- It cannot directly call triggers
- WRITETEXT statements do not allow a trigger

Advantages of Triggers:-

Triggers are useful for auditing data changes or auditing database as well as managing business rules.

Below are some examples:

- Triggers can be used to enforce referential integrity (For example you may not be able to apply foreign keys)
- Can access both new values and old values in the database when going to do any insert, update or delete

Disadvantages of Triggers

- Triggers hide database operations.

•For example when debugging a stored procedure, it's possible to not be aware that a trigger is on a table being checked for data changes
•Executing triggers can affect the performance of a bulk import operation.
Solution for Best Programming Practice
•Do not use triggers unnecessarily, if using triggers use them to resolve a specific situation
•Where possible, replace a trigger operation with a stored procedure or another kind of operation
•Do not write lengthy triggers as they can increase transaction duration; and also reduce the performance of data insert, update and delete operations as the trigger is fired every time the operation occurs.

Q. Explain triggers?

A special kind of "stored procedure" that goes into effect when you modify data.
- In a specified table using one or more data modification operations: UPDATE, DELETE, INSERT.
- Can query other tables and can include complex SQL statements

They are primarily useful for enforcing complex business rules or requirements
- you could control whether to allow an order to be inserted based on a customer's current account status useful for enforcing referential integrity, which preserves the defined relationships between tables when you add, update, or delete the rows in those tables
- A state in which all foreign key values in a database are valid

Triggers are automatic
- they are activated immediately after any modification to the table's data, such as a manual entry or an application action

Can cascade change trough related tables in a database
- you can write a delete trigger on the title_id column of the titles table to cause a deletion of matching rows in other tables
- can enforce restrictions that are more complex than those defined with check constraints

Defines which data value are acceptable in a column

❖ **INSERT Trigger**
❖ When a new row is inserted in the requisition table, the value of tax column should be less then salary column
❖ Ensure that this user-defined data integrity requirement is implemented
❖ Task List
- Identify the object that can maintain user defined data integrity
- Draft statement to create an INSERT trigger
- Create trigger in database
- Check the existence of trigger in the database
- Insert a row in the requisition table and verify that trigger is working

❖ Identify the object that can maintain user defined-data integrity
➢ A trigger is a block of code that constitutes a set of T-SQL statements that are activated in response to certain actions
➢ Characteristic trigger :
- It is fired automatically by DBMS when any data modification statement is issued
- It cannot be explicitly invoked or executed, as in the case of the stored procedures

❖ Identify …
➢ It prevents incorrect, unauthorized or inconsistent changes in data
➢ It cannot return data to the user
❖ Result
➢ A trigger can be used to maintain data integrity

Q. Which statement is used to create triggers?

- **Creating trigger**
  - Syntax

```
CREATE TRIGGER trigger_name
ON table_name
[WITH ENCRYPTION]
FOR [INSERT | DELETE | UPDATE]
AS sql_statements
```

- **Magic tables**
  - Whenever a trigger fires in response to the INSERT, DELETE or UPDATE statement, tw special table are created
  - These are the inserted and the deleted tables
  - Inserted table contains a copy of all records that are inserted in the trigger table
  - The delete table contains all records that have been deleted from trigger table
  - Whenever any update takes place, the trigger uses both the inserted and the deleted tables
- An INSERT Trigger is fired whenever an attempt is made to insert a row in the trigger table
- When an INSERT statement is issued, a new row is added to both trigger and the inserted tables
- Action :
  - The table on which the trigger has to be created is requisition
  - The trigger has to be of insert type
  - The name of trigger can be trginsertRequisiton
  - Write the batch statements

```
CREATE TRIGGER trgInsertRequisition
ON Requisition FOR insert AS
DECLARE @vacancyreported int
DECLARE @actualvacancy int
SELECT @actualvacancy = iBudgetedStrength –
    iCurrentStrength
FROM Position Join Inserted on Position.cPositionCode
    = Inserted.cPositionCode
```

- Create Trigger in the database
  - Action :
  - Type the drafted Code in the Query Analyzer window
  - Press F5 to execute the code
  - Check the existence of trigger in the database
  - Action :
  - Sp_help trgInsertRequisition
  - Insert a row in the Requisition table and verify that the trigger is working
  - Action :
  - INSERT Requisition
  - VALUES ('000003','0001',getdate(),getdate() + 7, '0001', 'North', 20)

Q. When is a DELETE Trigger fired?

- **DELETE Trigger**

- Create trigger to disable deleting rows from the ContractRecruiter table
- Task List
  - Draft statement to create a delete trigger
  - Create the trigger in the database
  - Check the existence of the trigger in the database
  - Delete a row from the Contractrecruiter table to verify the trigger
- A DELETE Trigger is fired whenever an attempt is made to delete rows from the trigger table
- There are three ways of implementing referential integrity using a DELETE trigger. These are :
  - The cascade method
  - The restrict method
  - The nulify method
- Result :

- The table on which the trigger is to be created is ContractRecruiter
- The trigger is a DELETE Trigger
- The name of the trigger is trgDeleteContractRecruiter
- The batch statements are :
  - CREATE TRIGGER trgDeleteContractRecruiter
  - ON ContractRecruiter FOR delete
  - AS
  - PRINT 'Deletion of Contract Recuiters is not allowed'
  - ROLLBACK TRANSACTION RETURN

  - Create trigger in the database
  - Type drafted code in the Query analyzer window
  - Press F5 to execute the code
- Check the existence of the trigger in the database
- Sp_help trgDeleteContractRecruiter
  - Delete a row from the ContractRecruiter table to verify the trigger
  - Action :

➢ Execute the following statement :
  - DELETE ContractRecruiter
  - WHERE cContractRecruiterCode = '000001'

➢ When this command is executed, the trigger is would be fired and it would prevent the deletion of rows from the ContractRecruiter table.

Q. Explain Nested triggers?

- A trigger that contains data modification logic within itself is called a nested trigger.
- Use the nested triggers option to control whether an AFTER trigger can cascade; that is, perform an action that initiates another trigger, which initiates another trigger, and so on. When nested triggers is set to 0, AFTER triggers cannot cascade. When nested triggers is set to 1 (the
- default), AFTER triggers can cascade to as many as 32 levels. INSTEAD OF triggers can be nested regardless of the setting of this option.
- If nested triggers are enabled, a trigger that changes a table on which there is another trigger fires the second trigger, which can in turn fire a third trigger, and so forth. If any trigger in the chain sets off an infinite loop, the nesting level is exceeded and the trigger aborts. You can use nested triggers to perform useful housekeeping functions such as storing a backup copy of rows affected by a previous trigger.

➢ **UPDATE Trigger**
  - Create a trigger so that the average siPercentageCharge attribut of the ContractRecruiter table should not be more than 11 when the value of siPercentageCharge is increased for any ContractRecruiter
  - Task List

- Draft statements to create an update trigger
- Create trigger in the database
- Check the existence of the trigger in the database
- Update siPercentageCharge of the ContractRecruiter table and verify that the average does not exceed the required value
- Draft statements to create an UPDATE trigger
- **The update trigger**
- Fired whenever there is a modification to the trigger table
- Result :
- The table on which the trigger is to be created is ContractRecruiter
- The trigger is an UPDATE trigger
- The name of the trigger is trgUpdateContractRecruiter

```
CREATE TRIGGER
    trgUpdateContractRecruiter
ON ContractRecruiter FOR UPDATE
AS
DECLARE @AvgPercentageCharge int
SELECT @AvgPercentageCharge =
    avg(siPercentageCharge)
FROM ContractRecruiter
```

- Create the trigger in the database
- In the query analyzer window, type the draft code
- Press F5 to execute the code
- Check the existence of the trigger in the database
- Sp_help trgUpdateContractRecruiter
- Update siPercentageCharge of the ContractRecruiter table and verify that the average does not exceed the required value

---

Q. Which statement is used to recreate a trigger?

➢ **Modifying the trigger**

- **Task List**
  - Draft the command to modify the trigger
  - Create the trigger in the database
  - Check that the trigger has been modified in the database
  - Insert a row in the requisition table and verify that the trigger is working

➢ **The ALTER TRIGGER Command**

- The contents of a trigger can be modified by:
  - Dropping the trigger and recreating it
  - Using the ALTER TRIGGER statement
- It is advisable to drop a trigger and recreate it, if the objects being referenced by it are renamed
- **Syntax:**

```
ALTER TRIGGER trigger_name
ON table_name
[WITH ENCRYPTION]
FOR [INSERT | DELETE | UPDATE]
```

AS sql_statements

- Action :
  - The table on which the trigger had been created is Requisiton
  - Determine the type of the trigger
  - The name of the trigger to be modified is trgInsertreRequisiton

- Check that the trigger has been modified in the database
  - Sp_helptext trgInsertRequisiton
- Insert a row in Requisition table and verify that the trigger is working

➢ **Dropping the Trigger**

- The DELETE trigger named trgDeleteContractRecruiter needs to be removed, as it is no longer required
- **Task List**
  - Draft the command to delete the trigger
  - Execute the command
  - Verify that the trigger has been removed
- The DROP Trigger command is used to delete a trigger from the database
  - DROP TRIGGER trigger_name[…,n]
- Action:
- The command to delete the trigger would be:
  - DROP TRIGGER trgDeleteContractRecruiter
- Execute the command
- Action:
  - In the query……..
  - Verify that the trigger has been removed
  - Sp_help trgDeleteContractRecruiter

➢ The above command will give an error message as the trigger has been removed

---

Q. Explain Enforcing Data Integrity Through Triggers?

➢ **Enforcing Data Integrity Through Triggers.**

- A trigger can be used to enforce business rules and data integrity in the following ways:
- If changes are made to the master table, then the same changes are cascaded to all the dependent tables
- If some changes violate referential integrity, then all such changes are rejected, thereby canceling any attempt to modify data in the database
- It allows very complex restriction to be enforced
- It can perform a particular action, depending on the outcome of the modifications that have been made to the tables
- **Multiple triggers**
- SQL Server allows multiple triggers to be defined on a given table. This implies that a single DML statement may fire two or more triggers. The triggers are fired in the order of creation

- It is very important for data in a database to be accurate and correct. There are various methods to ensure this, but triggers have their own significance in maintaining the integrity and consistency of data.

  A trigger can be used to enforce business rules and data integrity in the following ways:
  If changes are made to the master table, then the same changes will be cascaded to all the dependent tables.
  Suppose you have to delete a record from the Titles table, then all the records in the dependent table, TitleAuthor should also be deleted to maintain data integrity.
  Example

```
CREATE TRIGGER  trgDeleteTitle
ON Titles
FOR DELETE
AS
 DELETE TitleAuthor
FROM TitleAuthor  t  JOIN Delete  d
ON t.Title_ID = D.Title_ID
```

If some changes violate referential integrity, then all such changes are rejected, thereby cancelling any a attempt to modify data in the database.

It allows very complex restrictions to be enforced.

It can perform a particular action, depending on the modification that have been made to the table.

- A trigger can be used to ensure and enforce business rules and data integrity. Business rules refer to the policies of an organization, which ensure that its business runs smoothly. Data integrity refers to the accuracy and reliability of data.

Quick Revision

- A trigger is a stored procedure that goes into effect when you insert, delete, or update data in a table.
- **Triggers are automatic**: they are activated immediately after any modification to the table's data, such as a manual entry or an application action.
- **Trigger** Defines which data value are acceptable in a column
- Whenever a trigger fires in response to the INSERT, DELETE or UPDATE statement, two special table are created These are the inserted and the deleted tables.
- A trigger is a block of code that constitutes a set of T-SQL statements that are activated in response to certain actions.
- A trigger can be used to enforce business rules and data integrity in the following ways:
  - If changes are made to the master table, then the same changes are cascaded to all the dependent tables
  - If some changes violate referential integrity, then all such changes are rejected, thereby canceling any attempt to modify data in the database
  - It allows very complex restriction to be enforced
  - It can perform a particular action, depending on the outcome of the modifications that have been made to the tables
- A DELETE Trigger is fired whenever an attempt is made to delete rows from the trigger table

CHAPTER XII: TRANSACTION MANAGEMENT

TRANSACTION

•A transaction is a logical unit of work that contains one or more SQL statements. A transaction is an atomic unit. The effects of all the SQL statements in a transaction can be either all committed(applied to the database) or all rolled back(undone from the database).
•A transaction begins with the first executable SQL statement.
•A transaction ends when it is committed or rolled back, either explicitly with a COMMIT or ROLLBACK statement or implicitly when a DDL statement is issued.
•To illustrate the concept of a transaction, consider a banking database. When a bank customer transfers money from a savings account to a checking account, the transaction can consist of three separate operation:
i. Decrement the savings account
ii. Increment the checking account
iii.Record the transaction in the transaction journal
EXAMPLE:
To illustrate Banking transaction:

Transaction Begins

```
UPDATE savings_accounts
     SET balance = balance - 500
     WHERE account = 3209;
```
— Decrement Savings Account

```
UPDATE checking_accounts
     SET balance = balance + 500
     WHERE account = 3208;
```
— Increment Checking Account

```
INSERT INTO journal VALUES
     (journal_seq.NEXTVAL, '1B'
     3209, 3208, 500);
```
— Record in Transaction Journal

```
COMMIT WORK;
```
— End Transaction

Transaction Ends

PROPERTIES OF TRANSACTION:

Four properties of Transaction:
(ACID PROPERTIES)
1.Atomicity= all changes are made (commit), or none(rollback).
2.Consistency= transaction won't violate declared system integrity constraints
3.Isolation = results independent of concurrent transactions.
4.Durability= committed changes survive various classes of hardware failure
ATOMICITY
•All-or-nothing, no partial results.
•An event either happens and is committed or fails and is rolled back.

•EXAMPLE: In a money transfer, debit one account, credit the other. Either both debiting and crediting.
•If a transaction ends, we say its commits, otherwise it aborts.
•Transactions can be incomplete for three reasons:
1.It can be aborted by the DBMS,
2.A system crash.
3. The transaction aborts itself.
•When a transaction does not commit, its partial effects should be undone.
•Users can then forget about dealing with incomplete transactions.
•But if it is committed it should be durable
•The DBMS uses a log to ensure that incomplete transactions can be undone, if necessary.

CONSISTENCY

•If the database is in a consistent state before the execution of the transaction, the database remains consistent after the execution of the transaction.
Example:
Transaction T1 transfers $100 from Account A to Account B. Both Account A and Account B contains $500 each before the transaction.
Transaction T1
Read (A)
A=A-100
Write (A)
Read (B)
B=B+10
Consistency Constraint
Before Transaction execution Sum = A + B
Sum = 500 + 500
Sum = 1000
After Transaction execution Sum = A + B
Sum = 400 + 600
Sum = 1000
Before the execution of transaction and after the execution of transaction SUM must be equal.

ISOLATION

•Isolation requires that multiple transactions occurring at the same time not impact each other's execution.
•Example, if Joe issues a transaction against a database at the same time that Mary issues a different transaction; both transactions should operate on the database in an isolated manner.
•The database should either perform Joe's entire transaction before executing Mary's or vice -versa.
•This prevents Joe's transaction from reading intermediate data produced as a side effect of part of Mary's transaction that will not eventually be committed to the database.
•Note that the isolation property does not ensure which transaction will execute first, merely that they will not interfere with each other.

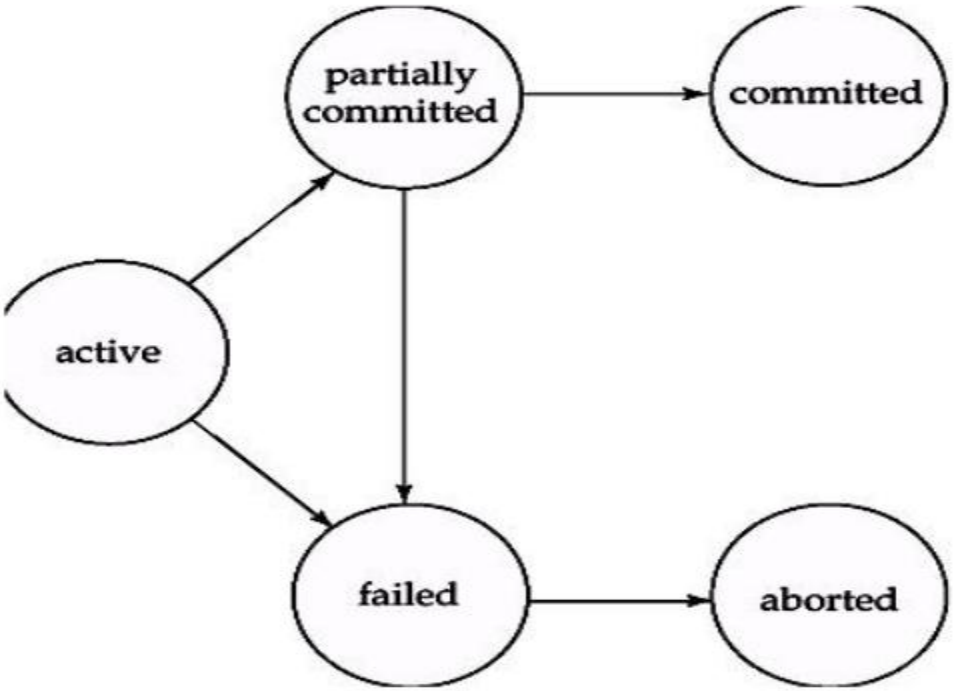
DURABILITY

•Durability ensures that any transaction committed to the database will not be lost.
•Durability is ensured through the use of database backups and transaction logs that facilitate the restoration of committed transactions in spite of any subsequent software or hardware failures.

TRANSACTION STATE DIAGRAM

The following are the different states in transaction processing in a Database System.

1.Active
2.Partially Committed
3.Failed
4.Aborted
5.Committed



1.Active
This is the initial state. The transaction stay in this state while it is executing.
2.Partially Committed
This is the state after the final statement of the transaction is executed.
3.Failed
After the discovery that normal execution can no longer proceed.
4.Aborted
The state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.
5.Committed
The state after successful completion of the transaction. We cannot abort or rollback a committed transaction.

TRANSACTION SCHEDULE

When multiple transactions are executing concurrently, then the order of execution of operations from the various transactions is known as schedule.
Serial Schedule Non- Serial Schedule
Serial Schedule
Transactions are executed one by one without any interleaved operations from other transactions.
Non-Serial Schedule
A schedule where the operations from a set of concurrent transactions are interleaved.

SERIALIZABILITY

What is Serializability?

A given non serial schedule of n transactions is serializable if it is equivalent to some serial schedule. i.e. this non serial schedule produce the same result as of the serial schedule.
Then the given non serial schedule is said to be serializable.
A schedule that is not serializable is called a non-serializable.
Non-Serial Schedule Classification
Serializable
Not Serializable
Recoverable
Non Recoverable
Serializable Schedule Classification
Conflict Serializable
View Serializable
Conflict
Serializable Schedule If a schedule S can be transformed into a schedule S' by a series of swaps of non-conflicting instruction then we say that S and S' are conflict equivalent.
A schedule S is called conflict serializable if it is conflict equivalent to a serial schedule.
View Serializable Schedule
All conflict serializable schedule are view serializable.
But there are view serializable schedule that are not conflict serializable.
A schedule S is a view serializable if it is view equivalent to a serial schedule.

Recoverable Schedule Classification

Cascade
Cascade less
To recover from the failure of a transaction Ti, we may have to rollback several transactions.
This phenomenon in which a single transaction failure leads to a series of transaction roll back is called cascading roll back.
Avoid cascading roll back by not allowing reading uncommitted data.
But this lead to a serial schedule.

Q. Explain Lock-Based Concurrency Control?

- LOCK-BASED CONCURRENCY CONTROL:

A DBMS must be able to ensure that only serializable, recoverable schedules are allowed and that no actions of committed transactions are lost while undoing aborted transactions. A DBMS typically uses a locking protocol to achieve this. A lock is a small bookkeeping object associated with a database object. A locking protocol is a set of rules to be followed by each transaction (and enforced by the DBMS) to ensure that, even though actions of several transactions might be interleaved, the net effect is identical to executing all transactions in some serial order. Different locking protocols use different types of locks, such as shared locks or exclusive locks, as we see next, when we discuss the Strict 2PL protocol.

- Strict Two-Phase Locking (Strict 2PL)

The most widely used locking protocol, called Strict Two-Phase Locking, or Strict 2PL, has two rules. The first rule is

1. If a transaction T wants to read (respectively, modify) an object, it first requests a shared (respectively, exclusive) lock on the object.

Of course, a transaction that has an exclusive lock can also read the object; an additional shared lock is not required. A transaction that requests a lock is suspended until the DBMS is able to grant it the requested lock. The DBMS keeps track of the locks it has granted and ensures that if a transaction holds an exclusive lock on an object, no other transaction holds a shared or exclusive lock on the same object. The second rule in Strict 2PL is

2. All locks held by a transaction are released when the transaction is completed.

Requests to acquire and release locks can be automatically inserted into transactions by the DBMS; users need not worry about these details.

In effect, the locking protocol allows only 'safe' interleaving's of transactions. If two transactions access completely independent parts of the database, they concurrently obtain the locks they need and proceed merrily on their ways. On the other hand, if two transactions access the same object, and one wants to modify it, their actions are effectively ordered serially—all actions of one of these transactions (the one that gets the lock on the common object first) are completed before (this lock is released and) the other transaction can proceed.

We denote the action of a transaction T requesting a shared (respectively, exclusive) lock on object O as $S_T(O)$ (respectively, $X_T(O)$) and omit the subscript denoting the transaction when it is clear from the context. As an example, consider the schedule shown in Figure 16.4. This interleaving could result in a state that cannot result from any serial execution of the three transactions. For instance, T1 could change A from 10 to 20, then T2 (which reads the value 20 for A) could change B from 100 to 200, and then T1 would read the value 200 for B. If run serially, either T1 or T2 would execute first, and read the values 10 for A and 100 for B: Clearly, the interleaved execution is not equivalent to either serial execution.

If the Strict 2PL protocol is used, such interleaving is disallowed. Let us see why. Assuming that the transactions proceed at the same relative speed as before, Ti would obtain an exclusive lock on A first and then read and write A (Figure 16.6). Then, T2 would request a lock on A. However, this request

$T1$	$T2$
$X(A)$	
$R(A)$	
$W(A)$	

Figure 16.6 Schedule Illustrating Strict 2PL

cannot be granted until T1 releases its exclusive lock on A, and the DBMS therefore suspends T2. Ti now proceeds to obtain an exclusive lock on B, reads and writes B, then finally commits, at which time its locks are released. T2's lock request is now granted, and it proceeds. In this example the locking protocol results in a serial execution of the two transactions, shown in Figure 16.7.

$T1$	$T2$
$X(A)$	
$R(A)$	
$W(A)$	
$X(B)$	
$R(B)$	
$W(B)$	
Commit	
	$X(A)$
	$R(A)$
	$W(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Commit

Figure 16.7 Schedule Illustrating Strict 2PL with Serial Execution

In general, however, the actions of different transactions could be interleaved. As an example, consider the interleaving of two transactions shown in Figure 16.8, which is permitted by the Strict 2PL protocol.

It can be shown that the Strict 2PL algorithm allows only serializable schedules. None of the anomalies discussed in Section 16.3.3 can arise if the DBMS implements Strict 2PL.

$T1$	$T2$
$S(A)$	
$R(A)$	
	$S(A)$
	$R(A)$
	$X(B)$
	$R(B)$
	$W(B)$
	Commit
$X(C)$	
$R(C)$	
$W(C)$	
Commit	

Figure 16.8 Schedule Following Strict 2PL with Interleaved Actions

- Deadlocks

Consider the following example. Transaction T1 sets an exclusive lock on object A, T2 sets an exclusive lock on B, T1 requests an exclusive lock on B and is queued, and T2 requests an exclusive lock on A and is queued. Now, T1 is waiting for T2 to release its lock and T2 is waiting for T1 to release its lock. Such a cycle of transactions waiting for locks to be released is called a deadlock. Clearly, these two transactions will make no further progress. Worse, they hold locks that may be required by other transactions. The DBMS

must either prevent or detect (and resolve) such deadlock situations; the common approach is to detect and resolve deadlocks.

A simple way to identify deadlocks is to use a timeout mechanism. If a transaction has been waiting too long for a lock, we can assume (pessimistically) that it is in a deadlock cycle and abort it.

Quick Revision

* A DBMS must ensure four important properties of transactions to maintain data in the face of concurrent access and system failures.
* Users are responsible for ensuring transaction consistency. That is, the user who submits a transaction must ensure that, when run to completion by itself against a 'consistent' database instance, the transaction will leave the database in a 'consistent' state.
* A transaction is seen by the DBMS as a series, or list, of actions. The actions that can be executed by a transaction include reads and writes of database objects.
* Transactions interact with each other only via database read and write operations; for example, they are not allowed to exchange messages.
* A serializable schedule over a set S of committed transactions is a schedule whose effect on any consistent database instance is guaranteed to be identical to that of some complete serial schedule over S.
* A schedule is conflict serializable if it is conflict equivalent to some serial schedule. Every conflict serializable schedule is serializable, if we assume that the set of items in the database does not grow or shrink; that is, values can be modified but items are not added or deleted.
* Conflict serializability is sufficient but not necessary for serializability. A more general sufficient condition is view serializability. Two schedules Si and S2 over the same set of transactions—any transaction that appears in either Si or S2 must also appear in the other—are view equivalent under these conditions:
    1. If Ti reads the initial value of object A in Si, it must also read the initial value of A in S2.
    2. If Ti reads a value of A written by Tj in Si, it must also read the value of A written by Tj in S2.
    3. For each data object A, the transaction (if any) that performs the final write on A in Si must also perform the final write on A in S2.

CHAPTER XIII: DATABASE RECOVERY

In this section, we consider how locking protocols guarantee some important properties of schedules; namely, serializability and recoverability. Two schedules are said to be conflict equivalent if they involve the (same set of) actions of the same transactions and they order every pair of conflicting actions of two committed transactions in the same way.

As we saw in Section 16.3.3, two actions conflict if they operate on the same data object and at least one of them is a write. The outcome of a schedule depends only on the order of conflicting operations; we can interchange any pair of nonconflicting operations without altering the effect of the schedule on the database. If two schedules are conflict equivalent, it is easy to see that they have the same effect on a database. Indeed, because they order all pairs of conflicting operations in the same way, we can obtain one of them from the other by repeatedly swapping pairs of nonconflicting actions, that is, by swapping pairs of actions whose relative order does not alter the outcome.

Q. Define conflict serializabilty and Precedence graphs and test for conflict seralizability. Examples related to them?

A schedule is conflict serializable if it is conflict equivalent to some serial schedule. Every conflict serializable schedule is serializable, if we assume that the set of items in the database does not grow or shrink; that is, values can be modified but items are not added or deleted. We make this assumption for now and consider its consequences in Section 17.5.1. However, some serializable schedules are not conflict serializable, as illustrated in Figure 17.1. This schedule is equivalent to executing the transactions serially in the order Ti, T2,

$T1$	$T2$	$T3$
$R(A)$		
	$W(A)$	
	Commit	
$W(A)$		
Commit		
		$W(A)$
		Commit

Figure 17.1 Serializable Schedule That Is Not Conflict Serializable

T3, but it is not conflict equivalent to this serial schedule because the writes of Ti and T2 are ordered differently.

It is useful to capture all potential conflicts between the transactions in a schedule in a precedence graph, also called a serializability graph. The precedence graph for a schedule S contains:

- A node for each committed transaction in S.
- An arc from Ti to Tj if an action of Ti precedes and conflicts with one of Tj's actions.

The precedence graphs for the schedules shown in Figures 16.7, 16.8, and 17.1 are shown in Figure 17.2 (parts a, b, and c, respectively).

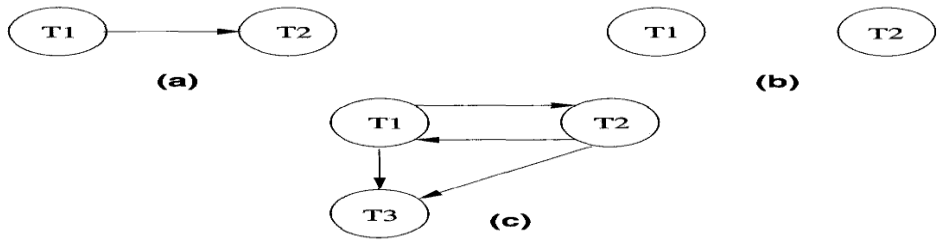


Figure 17.2 Examples of Precedence Graphs

The Strict 2PL protocol (introduced in Section 16.4) allows only conflict serializable schedules, as is seen from the following two results:

1. A schedule S is conflict serializable if and only if its precedence graph is acyclic. (An equivalent serial schedule in this case is given by any topological sort over the precedence graph.)

2. Strict 2PL ensures that the precedence graph for any schedule that it allows is acyclic.

A widely studied variant of Strict 2PL, called Two-Phase Locking (2PL), relaxes the second rule of Strict 2PL to allow transactions to release locks before the end, that is, before the commit or abort action. For 2PL, the second rule is replaced by the following rule:

(2PL) (2) A transaction cannot request additional locks once it releases any lock.

Thus, every transaction has a 'growing' phase in which it acquires locks, followed by a 'shrinking' phase in which it releases locks.

It can be shown that even nonstrict 2PL ensures acyclicity of the precedence graph and therefore allows only conflict serializable schedules. Intuitively, an equivalent serial order of transactions is given by the order in which transactions enter their shrinking phase: If T2 reads or writes an object written by Ti, Ti must have released its lock on the object before T2 requested a lock on this object. Thus, Ti precedes T2. (A similar argument shows that Ti precedes T2 if T2 writes an object previously read by Ti. A formal proof of the claim would have to show that there is no cycle of transactions that 'precede' each other by this argument.)

A schedule is said to be strict if a value written by a transaction T is not read or overwritten by other transactions until T either aborts or commits. Strict schedules are recoverable, do not require cascading aborts, and actions of aborted transactions can be undone by restoring the original values of modified objects. (See the last example in Section 16.3.4.) Strict 2PL improves on 2PL by guaranteeing that every allowed schedule is strict in addition to being conflict serializable. The reason is that when a transaction T writes an object under Strict 2PL, it holds the (exclusive) lock until it commits or aborts. Thus, no other transaction can see or modify this object until T is complete.

The reader is invited to revisit the examples in Section 16.3.3 to see how the corresponding schedules are disallowed by Strict 2PL and 2PL. Similarly, it would be instructive to work out

how the schedules for the examples in Section 16.3.4 are disallowed by Strict 2PL but not by 2PL.

Q. Why conflict seralizability is not necessary for seralizability? Example.

- View Serializability

Conflict serializability is sufficient but not necessary for serializability. A more general sufficient condition is view serializability. Two schedules Si and S2 over the same set of transactions—any transaction that appears in either Si or S2 must also appear in the other—are view equivalent under these conditions:

1. If Ti reads the initial value of object A in Si, it must also read the initial value of A in S2.

2. If Ti reads a value of A written by Tj in Si, it must also read the value of A written by Tj in S2.

3. For each data object A, the transaction (if any) that performs the final write on A in Si must also perform the final write on A in S2.

A schedule is view serializable if it is view equivalent to some serial schedule. Every conflict serializable schedule is view serializable, although the converse is not true. For example, the schedule shown in Figure i7.i is view serializable, although it is not conflict serializable. Incidentally, note that this example contains blind writes. This is not a coincidence; it can be shown that any view serializable schedule that is not conflict serializable contains a blind write.

As we saw in Section i7.i, efficient locking protocols allow us to ensure that only conflict serializable schedules are allowed. Enforcing or testing view serializability turns out to be much more expensive, and the concept therefore has little practical use, although it increases our understanding of serializability.

Q. What is a lock? What is a need? What are the different types of locks (Shared and exclusive locks)?

- <u>INTRODUCTION TO LOCK MANAGEMENT:</u>

The part of the DBMS that keeps track of the locks issued to transactions is called the lock manager. The lock manager maintains a lock table, which is a hash table with the data object identifier as the key. The DBMS also maintains a descriptive entry for each transaction in a transaction table, and among other things, the entry contains a pointer to a list of locks held by the transaction. This list is checked before requesting a lock, to ensure that a transaction does not request the same lock twice.

A lock table entry for an object—which can be a page, a record, and so on, depending on the DBMS—contains the following information: the number of transactions currently holding a lock on the object (this can be more than one if the object is locked in shared mode), the nature of the lock (shared or exclusive), and a pointer to a queue of lock requests.

- Implementing Lock and Unlock Requests

According to the Strict 2PL protocol, before a transaction T reads or writes a database object 0, it must obtain a shared or exclusive lock on 0 and must hold on to the lock until it commits or aborts. When a transaction needs a lock on an object, it issues a lock request to the lock manager:

1. If a shared lock is requested, the queue of requests is empty, and the object is not currently locked in exclusive mode, the lock manager grants the lock and updates the lock table entry for the object (indicating that the object is locked in shared mode, and incrementing the number of transactions holding a lock by one).

2. If an exclusive lock is requested and no transaction currently holds a lock on the object (which also implies the queue of requests is empty), the lock manager grants the lock and updates the lock table entry.

3. Otherwise, the requested lock cannot be immediately granted, and the lock request is added to the queue of lock requests for this object. The transaction requesting the lock is suspended.

When a transaction aborts or commits, it releases all its locks. When a lock on an object is released, the lock manager updates the lock table entry for the object and examines the lock request at the head of the queue for this object. If this request can now be granted, the transaction that made the request is woken up and given the lock. Indeed, if several requests for a shared lock on the object are at the front of the queue, all of these requests can now be granted together.

Note that if Ti has a shared lock on 0 and T2 requests an exclusive lock, T2's request is queued. Now, if T3 requests a shared lock, its request enters the queue behind that of T2, even though the requested lock is compatible with the lock held by Ti. This rule ensures that T2 does not starve, that is, wait indefinitely while a stream of other transactions acquire shared locks and thereby prevent T2 from getting the exclusive lock for which it is waiting.

- Atomicity of Locking and Unlocking

The implementation of lock and unlock commands must ensure that these are atomic operations. To ensure atomicity of these operations when several instances of the lock manager code can execute concurrently, access to the lock table has to be guarded by an operating system synchronization mechanism such as a semaphore.

To understand why, suppose that a transaction requests an exclusive lock. The lock manager checks and finds that no other transaction holds a lock on the object and therefore decides to grant the request. But, in the meantime, another transaction might have requested and received a conflicting lock. To prevent this, the entire sequence of actions in a lock request call (checking to see if the request can be granted, updating the lock table, etc.) must be implemented as an atomic operation.

- Other Issues: Latches, Convoys

In addition to locks, which are held over a long duration, a DBMS also supports short-duration latches. Setting a latch before reading or writing a page ensures that the physical read or write operation is atomic; otherwise, two read/write operations might

conflict if the objects being locked do not correspond to disk pages (the units of I/O). Latches are unset immediately after the physical read or write operation is completed.

We concentrated thus far on how the DBMS schedules transactions based on their requests for locks. This interleaving interacts with the operating system's scheduling of processes' access to the CPU and can lead to a situation called a convoy, where most of the CPU cycles are spent on process switching. The problem is that a transaction T holding a heavily used lock may be suspended by the operating system. Until T is resumed, every other transaction that needs this lock is queued. Such queues, called convoys, can quickly become very long; a convoy, once formed, tends to be stable. Convoys are one of the drawbacks of building a DBMS on top of a general-purpose operating system with preemptive scheduling.

- LOCK CONVERSIONS:

A transaction may need to acquire an exclusive lock on an object for which it already holds a shared lock. For example, a SQL update statement could result in shared locks being set on each row in a table. If a row satisfies the condition (in the WHERE clause) for being updated, an exclusive lock must be obtained for that row.

Such a lock upgrade request must be handled specially by granting the exclusive lock immediately if no other transaction holds a shared lock on the object and inserting the request at the front of the queue otherwise. The rationale for favoring the transaction thus is that it already holds a shared lock on the object and queuing it behind another transaction that wants an exclusive lock on the same object causes both a deadlock. Unfortunately, while favoring lock upgrades helps, it does not prevent deadlocks caused by two conflicting upgrade requests. For example, if two transactions that hold a shared lock on an object both request an upgrade to an exclusive lock, this leads to a deadlock.

A better approach is to avoid the need for lock upgrades altogether by obtaining exclusive locks initially, and downgrading to a shared lock once it is clear that this is sufficient. In our example of an SQL update statement, rows in a table are locked in exclusive mode first. If a row does not satisfy the condition for being updated, the lock on the row is downgraded to a shared lock. Does the downgrade approach violate the 2PL requirement? On the surface, it does, because downgrading reduces the locking privileges held by a transaction, and the transaction may go on to acquire other locks. However, this is a special case, because the transaction did nothing but read the object that it downgraded, even though it conservatively obtained an exclusive lock. We can safely expand our definition of 2PL from Section 17.1 to allow lock downgrades in the growing phase, provided that the transaction has not modified the object.

The downgrade approach reduces concurrency by obtaining write locks in some cases where they are not required. On the whole, however, it improves throughput by reducing deadlocks. This approach is therefore widely used in current commercial systems. Concurrency can be increased by introducing a new kind of lock, called an update lock, that is compatible with shared locks but not other update and exclusive locks. By setting an update lock initially, rather than exclusive locks, we prevent conflicts with other read operations. Once we are sure we need not update the object, we can downgrade to a shared lock. If we need to update the object, we must first upgrade to an exclusive

lock. This upgrade does not lead to a deadlock because no other transaction can have an upgrade or exclusive lock on the object.

Q. Explain Deadlocks?

- DEALING WITH DEADLOCKS:

Deadlocks tend to be rare and typically involve very few transactions. In practice, therefore, database systems periodically check for deadlocks. When a transaction Ti is suspended because a lock that it requests cannot be granted, it must wait until all transactions Tj that currently hold conflicting locks release them. The lock manager maintains a structure called a waits-for graph to detect deadlock cycles. The nodes correspond to active tranactions, and there is an arc from Ti to Tj if (and only if) Ti is waiting for Tj to release a lock. The lock manager adds edges to this graph when it queues lock requests and removes edges when it grants lock requests.

Consider the schedule shown in Figure 17.3. The last step, shown below the line, creates a cycle in the waits-for graph. Figure 17.4 shows the waits-for graph before and after this step.

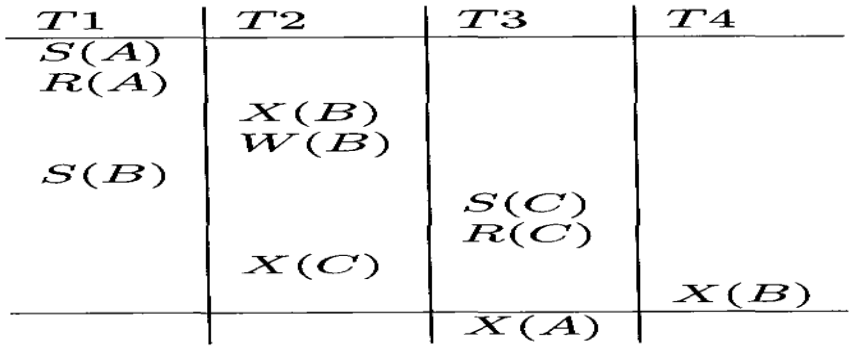


Figure 17.3 Schedule Illustrating Deadlock

Figure 17.4 Waits-for Graph Before and After Deadlock

Observe that the waits-for graph describes all active transactions, some of which eventually abort. If there is an edge from Ti to Tj in the waits-for graph, and both Ti and Tj eventually commit, there is an edge in the opposite direction (from Tj to Ti) in the precedence graph (which involves only committed transactions).

The waits-for graph is periodically checked for cycles, which indicate deadlock. A deadlock is resolved by aborting a transaction that is on a cycle and releasing its locks; this action allows some of the waiting transactions to proceed. The choice of which transaction to abort can be made using several criteria: the one with the fewest locks, the one that has done the least work, the one that is farthest from completion, and so on.

Further, a transaction might have been repeatedly restarted; if so, it should eventually be favored during deadlock detection and allowed to complete.

A simple alternative to maintaining a waits-for graph is to identify deadlocks through a timeout mechanism: If a transaction has been waiting too long for a lock, we assume (pessimistically) that it is in a deadlock cycle and abort it.

Q. Explain Deadlock prevention?

- Deadlock Prevention

Empirical results indicate that deadlocks are relatively infrequent, and detection-based schemes work well in practice. However, if there is a high level of contention for locks and therefore an increased likelihood of deadlocks, prevention- based schemes could perform better. We can prevent deadlocks by giving each transaction a priority and ensuring that lower-priority transactions are not allowed to wait for higher-priority transactions (or vice versa). One way to assign priorities is to give each transaction a timestamp when it starts up. The lower the timestamp, the higher is the transaction's priority; that is, the oldest transaction has the highest priority.

If a transaction Ti requests a lock and transaction Tj holds a conflicting lock, the lock manager can use one of the following two policies:

- Wait-die: If Ti has higher priority, it is allowed to wait; otherwise, it is aborted.
- Wound-wait: If Ti has higher priority, abort Tj; otherwise, Ti waits.

In the wait-die scheme, lower-priority transactions can never wait for higher priority transactions. In the wound-wait scheme, higher-priority transactions never wait for lower-priority transactions. Iii either case, no deadlock cycle develops.

A subtle point is that we must also ensure that no transaction is perennially aborted because it never has a sufficiently high priority. (Note that, in both schemes, the higher-priority transaction is never aborted.) When a transaction is aborted and restarted, it should be given the same timestamp it had originally. Reissuing timestamps in this way ensures that each transaction will eventually become the oldest transaction, and therefore the one with the highest priority, and will get all the locks it requires.

The wait-die scheme is no preemptive; only a transaction requesting a lock can be aborted. As a transaction grows older (and its priority increases), it tends to wait for more and more younger transactions. A younger transaction that conflicts with an older transaction may be repeatedly aborted (a disadvantage with respect to wound-wait), but on the other hand, a transaction that has all the locks it needs is never aborted for deadlock reasons (an advantage with respect to wound-wait, which is preemptive).

A variant of 2PL, called Conservative 2PL, can also prevent deadlocks. Under Conservative 2PL, a transaction obtains all the locks it will ever need when it begins, or blocks waiting for these locks to become available. This scheme ensures that there will be no deadlocks, and, perhaps more important, that a transaction that already holds some locks will not block waiting for other locks. If lock contention is heavy, Conservative 2PL can reduce the time that locks are held on average, because transactions that hold locks are

never blocked. The trade-off is that a transaction acquires locks earlier, and if lock contention is low, locks are held longer under Conservative 2PL. From a practical perspective, it is hard to know exactly what locks are needed ahead of time, and this approach leads to setting more locks than necessary. It also has higher overhead for setting locks because a transaction has to release all locks and try to obtain them all over if it fails to obtain even one lock that it needs. This approach is therefore not used in practice.

Q. Explain Concurrency control by time stamps and Thomos Write rule?

- Timestamp-Based Concurrency Control

In lock-based concurrency control, conflicting actions of different transactions are ordered by the order in which locks are obtained, and the lock protocol extends this ordering on actions to transactions, thereby ensuring serializability. In optimistic concurrency control, a timestamp ordering is imposed on transactions and validation checks that all conflicting actions occurred in the same order.

Timestamps can also be used in another way: Each transaction can be assigned a timestamp at startup, and we can ensure, at execution time, that if action a i of transaction T i conflicts with action aj of transaction T j , a i occurs before a j if TS(Ti) < TS(Tj). If an action violates this ordering, the transaction is aborted and restarted.

To implement this concurrency control scheme, every database object 0 is given a read timestamp RTS(0) and a write timestamp WTS(O). If transaction T wants to read object 0, and TS(T) < WTS(O), the order of this read with respect to the most recent write on 0 would violate the timestamp order between this transaction and the writer. Therefore, T is aborted and restarted with a new, larger timestamp. If TS(T) > WTS(0), T reads 0, and RTS(O) is set to the larger of RTS(0) and TS(T). (Note that a physical change—the change to RTS(O)—is written to disk and recorded in the log for recovery purposes, even on reads. This write operation is a significant overhead.)

Observe that if T is restarted with the same timestamp, it is guaranteed to be aborted again, due to the same conflict. Contrast this behavior with the use of timestamps in 2PL for deadlock prevention, where transactions are restarted with the same timestamp as before to avoid repeated restarts. This shows that the two uses of timestamps are quite different and should not be confused.

Next, consider what happens when transaction T wants to write object o:

1. If TS(T) < RTS(O), the write action conflicts with the most recent read action of o, and T is therefore aborted and restarted.

2. If TS(T) < WTS(0), a naive approach would be to abort T because its write action conflicts with the most recent write of o and is out of timestamp order. However, we can safely ignore such writes and continue. Ignoring outdated writes is called the Thomas Write Rule.

3. Otherwise, T writes o and WTS(O) is set to TS(T).

- The Thomas Write Rule

We now consider the justification for the Thomas Write Rule. If TS(T) < WTS(O), the current write action has, in effect, been made obsolete by the most recent write of 0, which follows the current write according to the times- tamp ordering. We can think of T's write action as if it had occurred immediately before the most recent write of 0 and was never read by anyone.

If the Thomas Write Rule is not used, that is, T is aborted in case (2), the timestamp protocol, like 2PL, allows only conflict serializable schedules. If the Thomas Write Rule is used, some schedules are permitted that are not conflict serializable, as illustrated by the schedule in Figure 17.6.2 Because T2's write follows T1's read and precedes T1's write of the same object, this schedule is not conflict serializable.

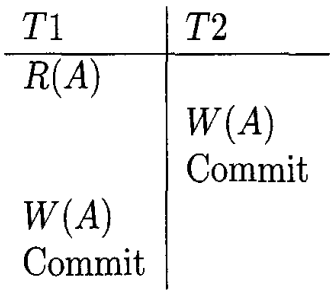


Figure 17.6 A Serializable Schedule That Is Not Conflict Serializable

The Thomas Write Rule relies on the observation that T2's write is never seen by any transaction and the schedule in Figure 17.6 is therefore equivalent to the serializable schedule obtained by deleting this write action, which is shown in Figure 17.7.

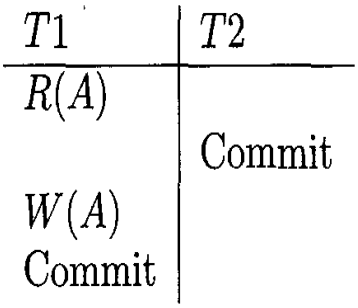


Figure 17.7 A Conflict Serializable Schedule

Quick Revision

* A schedule is conflict serializable if it is conflict equivalent to some serial schedule. Every conflict serializable schedule is serializable, if we assume that the set of items in the database does not grow or shrink; that is, values can be modified but items are not added or deleted.
* The precedence graph for a schedule S contains:
    - A node for each committed transaction in S.
    - An arc from Ti to Tj if an action of Ti precedes and conflicts with one of Tj's actions.
* The part of the DBMS that keeps track of the locks issued to transactions is called the lock manager. The lock manager maintains a lock table, which is a hash table with the data object identifier as the key.
* A schedule is said to be strict if a value written by a transaction T is not read or overwritten by other transactions until T either aborts or commits.
* If a shared lock is requested, the queue of requests is empty, and the object is not currently locked in exclusive mode, the lock manager grants the lock and updates the lock table entry for the object (indicating that the object is locked in shared mode, and incrementing the number of transactions holding a lock by one).
* If an exclusive lock is requested and no transaction currently holds a lock on the object (which also implies the queue of requests is empty), the lock manager grants the lock and updates the lock table entry.
* In lock-based concurrency control, conflicting actions of different transactions are ordered by the order in which locks are obtained, and the lock protocol extends this ordering on actions to transactions, thereby ensuring serializability.
* A transaction may need to acquire an exclusive lock on an object for which it already holds a shared lock.