DATABASE SYSTEMS
[R22A0553]

LECTURE NOTES

B.TECH
III YEAR–II SEM
(R22)
ACADEMIC YEAR (2024-25)

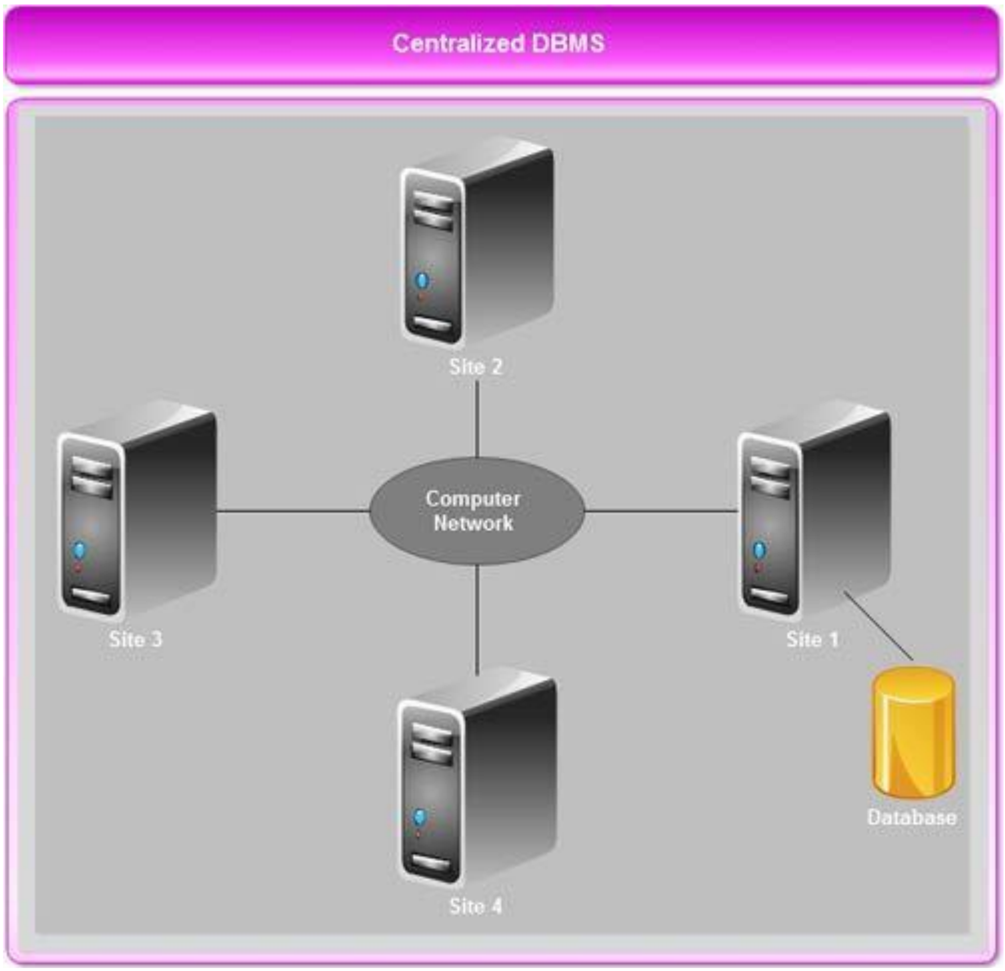
DEPARTMENT OF EEE

MALLA REDDY COLLEGE OF ENGINEERING&TECHNOLOGY
(Autonomous Institution–UGC, Govt. of India)
Recognized under 2(f) and 12(B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE-Accredited by NBA&NAAC– 'A' Grade-ISO9001:2015Certified)
Maisammaguda, Dhulapally (PostVia.Hakimpet), Secunderabad- 500100,Telangana.

DIGITAL NOTES ON

DATABASE SYSTEMS (R22A0553)

B.TECH III YEAR-II SEM (2024-25)



PREPARED BY

G NARESH

MALLAREDDYCOLLEGEOFENGINEERINGANDTECHNOLOGY

III Year B.Tech.EEE- IISem **L/T/P/C**
3/-/-/3

OPEN ELECTIVE - II (R22A0553)
DATABASE SYSTES

COURSE OBJECTIVES:

To understand the basic concepts and the applications of database systems

1) To Master the basics of SQL and construct queries using SQL
2) To understand the relational database design principles
3) To become familiar with the basic issues of transaction processing and concurrency control
4) To become familiar with database storage structures and access techniques

UNITI:
INTRODUCTION
Database:PurposeofDatabaseSystems,FileProcessingSystemVsDBMS,History,Characteristic- Three schema Architecture of a database, Functional components of a DBMS.DBMS Languages- Database users and DBA.

UNITII:
DATABASEDESIGN
ER Model: Objects, Attributes and its Type. Entity set and Relationship set-Design Issues of ER model-Constraints. Keys-primary key, Super key, candidate keys. Introduction to relational model-Tabular, Representation of Various ER Schemas. ER Diagram Notations- Goals of ER Diagram- Weak Entity Set- Views.

UNITIII:
STRUCTUREDQUERY LANGUAGE
SQL: Overview, The Form of Basic SQL Query -UNION, INTERSECT, and EXCEPT– join operations: equi join and non equi join-Nested queries - correlated and uncorrelated- Aggregate Functions- Null values.Views, Triggers.

UNITIV:
DEPENDENCIESANDNORMALFORMS
Importance of a good schema design,:- Problems encountered with bad schema designs, Motivation for normal forms- functional dependencies, -Armstrong's axioms for FD's- Closure of a set of FD's,- Minimal covers-Definitions of 1NF,2NF, 3NF and BCNF- Decompositions and desirable properties.

UNITV:
Transactions: Transaction concept, transaction state, System log, Commit point, Desirable PropertiesofaTransaction,concurrentexecutions,serializability,recoverability,implementation of isolation, transaction definition in SQL, Testing for serializability**,** Serializability by Locks- Locking Systems with Several Lock Modes- Concurrency Control by Timestamps, validation.

TEXTBOOKS:

1) AbrahamSilberschatz,HenryF.Korth,S.Sudarshan,‖DatabaseSystem Concepts‖,McGraw-Hill,6thEdition,2010.
2) FundamentalofDatabaseSystems,byElmasri,Navathe,Somayajulu,andGupta, Pearson Education.

REFERENCE BOOKS:

1) RaghuRamakrishnan,JohannesGehrke,—DatabaseManagementSystem‖,McGrawHill., 3rd Edition 2007.
2) Elmasri&Navathe,‖FundamentalsofDatabaseSystem,‖Addison-WesleyPublishing,5th Edition, 2008.
3) Date.C.J,—AnIntroductiontoDatabase‖,Addison-WesleyPubCo,8thEdition,2006.
4) Peterrob,Carlos Coronel, —Database Systems – Design, Implementation, and Management‖,9thEdition,ThomsonLearning,2009.

COURSEOUTCOMES:

1) Understand the basic concepts and the applications of data base systems
2) Master the basics of SQL and construct queries using SQL
3) Understand the relational data base design principles
4) Familiarize with the basic issues of transaction processing and concurrency control
5) Familiarize with database storage structures and access techniques

UNIT I

INTRODUCTION

DATABASE

Data:

Itisacollectionof information.

Thefactsthatcanberecordedandwhichhaveimplicitmeaningknownas'data'.

Example:

Customer-----

1.cname.

2.cno.

3.ccity.

Database:

- Itisacollectionofinterrelateddata.
- Thesecanbestoredintheformoftables.
- Adatabasecanbeofanysizeandvaryingcomplexity.
- Adatabasemaybegeneratedandmanipulatedmanuallyoritmaybecomputerized.

Example:

Customerdatabaseconsiststhefieldsascname,cno,and ccity

Cname	Cno	Ccity

DatabaseSystem:

It is computerized system, whose overall purpose is to maintain the information and to make that the information is available on demand.

Advantages:

1.Redundency can be reduced

2.Inconsistencycanbeavoided.

3.Datacan be shared

4.Standardscanbeenforced.

5.Securityrestrictionscanbeapplied. 6.Integrity can be maintained.

7.Datagatheringcanbepossible.

8.Requirementscanbebalanced.

A database in a DBMS could be viewed by lots of different people with different responsibilities.

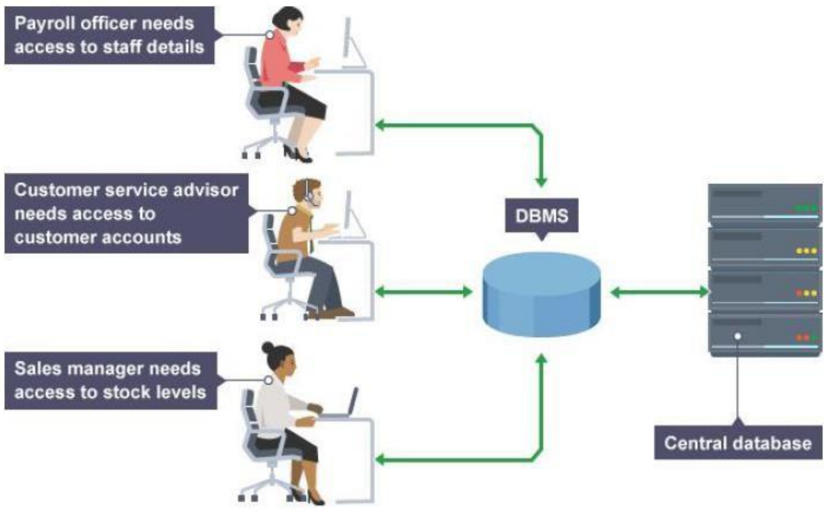


Figure:EmployeesareaccessingDatathrough DBMS

For example, within a company there are different departments, as well as customers, who each need to see different kinds of data. Each employee in the company will have different levels of access to the database with their own customized **front-end** application.

In a database, data is organized strictly in row and column format. The rows are called **Tuple** or **Record**. The data items within one row may belong to different data types. On the other hand, the columns are often called **Domain** or **Attribute**. All the data items within a single attribute are of the same data type.

WhatisManagement System?

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database.

The collection of data, usually referred to as the **database**, contains information relevant toan enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*. By **data,** we mean known facts that can be recorded and that have implicit meaning.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms forthe 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.

DatabaseManagementSystem(DBMS):

It is a collection of programs that enables user to create and maintain a database. In other words it is general-purpose software that provides the users with the processes of defining, constructing and manipulating the database for various applications.

Databases touch all aspects of our lives. Some of the major areas of application are asfollows:

1. Banking
2. Airlines
3. Universities
4. Manufacturingand selling
5. Humanresources

EnterpriseInformation

◦ *Sales*:Forcustomer,product,andpurchase information.

◦ *Accounting*:Forpayments,receipts,accountbalances,assetsandotheraccounting information.

◦ *Human resources*: For information about employees, salaries, payroll taxes, and benefits,and for generation of pay checks.

◦ *Manufacturing*:Formanagementofthesupplychainandfortrackingproductionofitemsin factories, inventories of items in warehouses and stores, and orders for items.

***Onlineretailers*:**Forsalesdatanotedaboveplusonlineordertracking,generationof recommendation lists, and maintenance of online product evaluations.

◦ *Banking*:Forcustomerinformation,accounts,loans,andbankingtransactions.

◦ *Credit card transactions*: For purchases on credit cards and generation of monthly statements.

◦ *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 online trading by customers and automated trading by the firm.

***Universities*:** For student information, course registrations, and grades (in addition tostandard enterprise information such as human resources and accounting).

***Airlines*:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.

***Telecommunication*:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

PurposeofDatabaseSystems

Database systems arose in response to early methods of computerized management of commercial data. As an example of such methods, typical of the 1960s, consider part of a university organization that, among other data, keeps information about all instructors, students, departments, and course offerings. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programsto:

- Addnewstudents,instructors,andcourses
- Registerstudentsforcoursesandgenerateclass rosters
- Assigngradestostudents,computegradepointaverages (GPA),andgeneratetranscripts

This typical **file-processing system** is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files. Before databasemanagement systems (DBMSs) were introduced, organizations usually stored information in such systems. Keeping organizational information in **a file-processing system** has a number of **major disadvantages**:

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 structuresandtheprograms maybewritteninseveralprogramminglanguages.Moreover,the same information may be duplicated in several places (files). For example, if a student has a double major (say, music and mathematics) the address and telephone number of that student may appear in a file that consists of student records of students in the Music department and in a file that consists of student records of students in the Mathematics department. 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.

Forexample,achangedstudentaddressmaybereflectedintheMusicdepartmentrecordsbut not elsewhere in the system.

Difficulty in accessing data: Suppose 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 systemdid 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.

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 thebalanceamountineachaccount.Supposealsothattheuniversityrequiresthattheaccount 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 newconstraintsareadded,itis difficulttochangetheprogramstoenforce them.Theproblem is compounded when constraints involve several data items from different files.

Atomicityproblems:Acomputersystem,likeanyotherdevice,issubjecttofailure. Inmany applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.

Concurrent-access anomalies. For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concurrent updates is possible and mayresultininconsistentdata.Considerdepartment*A*,withanaccountbalance of$10,000.Iftwo department clerks debit the account balance (by say $500 and $100, respectively) of department *A* at almost exactly the same time, the result of the concurrent executions may leave the budget in an incorrect (or inconsistent) state.

Suppose that the programs executing on behalf of each withdrawal read the old balance, reduce that value by the amount being withdrawn, and write the result back. If the two programsrunconcurrently,theymaybothreadthevalue$10,000,andwriteback$9500and $9900, respectively. Depending on which one writes the value last, the account balance of department *A* may contain either $9500 or $9900, rather than the correct value of $9400. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many differentapplication programs that have not been coordinated previously.

Security problems. Not every user of the database system should be able to access all the data. For example, in a university, payroll personnel need to see only that part of the database that has financial information. They do not need access to information about academic records. But, since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult. These difficulties, among others, prompted the development of database systems. In what follows, we shall see the concepts and algorithms that enable database systems to solve the problems with file processing systems.

Advantagesof DBMS:

Controlling of Redundancy: Data redundancy refers to the duplication of data (i.e storing same data multiple times). In a database system, by having a centralized database and centralized control of data by the DBA the unnecessary duplication of data is avoided. It also eliminates the extra time for processing the large volume of data. It results in saving the storage space.

Improved Data Sharing : DBMS allows a user to share the data in any number of application programs.

Data Integrity : Integrity means that the data in the database is accurate. Centralized control of the data helps in permitting the administrator to define integrity constraints to the data in the database. For example: in customer database we can can enforce an integrity that it must accept the customer only from Noida and Meerut city.

Security : Having complete authority over the operational data, enables the DBA in ensuring that the only mean of access to the database is through proper channels. The DBA can define authorization checks to be carried out whenever access to sensitive data is attempted.

Data Consistency : By eliminating data redundancy, we greatly reduce the opportunities for inconsistency.Forexample:isacustomeraddressisstoredonlyonce,wecannothave disagreement on the stored values. Also updating data values is greatly simplified when each valueisstoredinoneplaceonly.Finally,weavoidthewastedstoragethatresultsfrom redundant data storage.

EfficientDataAccess: Inadatabasesystem,thedataismanagedbytheDBMSandall access to the data is through the DBMS providing a key to effective data processing **Enforcements of Standards** : With the centralized of data, DBA can establish and enforcethe data standards which may include the naming conventions, data quality standards etc.

Data Independence : Ina database system, the database management system provides the interface
between the application programs and the data. When changes are made to the data representation,themetadataobtainedbytheDBMSischangedbuttheDBMSiscontinuesto provide the data to application program in the previously used way. The DBMs handles the task of transformation of data wherever necessary.

Reduced Application Development and Maintenance Time : DBMS supports many important functions that are common to many applications, accessing data stored in the DBMS, which facilitates the quick development of application

DisadvantagesofDBMS

- It is bit complex. Since it supports multiple functionality to give the user the best, the underlying software has become complex. The designers and developers should have thorough knowledge about the software to get the most out of it.
- Because of its complexity and functionality, it uses large amount of memory. It also needs large memory to run efficiently.
- DBMS system works on the centralized system, i.e.; all the users from all over the world access
- thisdatabase.Hence anyfailureoftheDBMS,will impactalltheusers.
- DBMS is generalized software, i.e.; it is written work on the entire systems rather specific one. Hence some of the application will run slow.

Peoplewhodealwithdatabases

Many persons are involved in the design, use and maintenance of any database. These persons can be classified into 2 types as below.

Actorsonthescene:

The people, whose jobs involve the day-to-day use of a database are called as 'Actors on the scene', listed as below.

1. DatabaseAdministrators(DBA):

The DBA is responsible for authorizing access to the database, forCoordinating and monitoring its useandforacquiringsoftwareandhardwareresourcesasneeded.Thesearethepeople,whomaintain and design the database daily.DBA is responsible for the following issues.

- **Designoftheconceptualandphysicalschemas:**
  The DBA is responsible for interacting with the users of the system to understand what datais to be stored in the DBMS and how it is likely to be used. The DBA creates the original schema by writing a set of definitions and isPermanently stored in the 'Data Dictionary'.
- **Securityand Authorization:**
  The DBA is responsible for ensuring the unauthorized data access is not permitted.The granting of different types of authorization allows the DBA to regulate which parts of the database various users can access.
- **StoragestructureandAccessmethoddefinition:**
  The DBA creates appropriate storage structures and access methodsby writing a set of definitions, which are translated by the DDL compiler.
- **Data Availability and Recovery from Failures:** The DBA must take steps to ensure that if thesystemfails,userscancontinuetoaccessasmuchoftheuncorrupteddataaspossible.The DBA also work to restore the data to consistent state.
- **Database Tuning:**
  The DBA is responsible for modifying the database to ensure adequatePerformance as requirementschange.Integrity Constraint Specification:The integrity constraints are kept in a special system structure that is consulted by the DBA whenever an update takes place in the system.

2. DatabaseDesigners:

Databasedesignersareresponsibleforidentifyingthedatatobestoredinthedatabaseandfor choosing appropriate structures to represent and store this data.

3. End Users:

Peoplewhowishtostoreandusedatainadatabase.

Endusersarethepeoplewhosejobsrequireaccesstothedatabaseforquerying,updatingand generating reports, listed as below.

- **CasualEnd users:**
  These people occasionally access the database, but they may need different information each time.
- **NaiveorParametricEndUsers:**
  Theirjobfunctionrevolvesaroundconstantlyqueryingandupdatingthedatabaseusing standard types of queries and updates.
- **SophisticatedEndUsers:**
  TheseincludeEngineers,Scientists,Businessanalystandothersfamiliarizetoimplement their applications to meet their complex requirements.
- **StandaloneEndusers:**

Thesepeoplemaintainpersonaldatabasesbyusingready-madeprogrampackagesthatprovide easy to use menu based interfaces.

4. SystemAnalyst:

Thesepeopledeterminetherequirementsofendusersanddevelopspecificationsfortransactions.

5. ApplicationProgrammers(Software Engineers):

Thesepeoplecantest,debug,documentandmaintainthespecifiedtransactions.

b.Workersbehindthe scene:

DatabaseDesignersandImplementers:

ThesepeoplewhodesignandimplementtheDBMSmodulesandinterfacesasasoftware package.

2. Tool Developers**:**

Includepersonswhodesignandimplementtoolsconsistingthepackagesfordesign,performance monitoring, and prototyping and test data generation.

3. Operatorsandmaintenancepersonnel:

Theserethesystemadministrationpersonnelwhoareresponsiblefortheactualrunningand maintenance of the hardware and software environment for the database system.

LEVELSOFDATAABSTRACTION

Thisisalsocalledas'TheThree-SchemaArchitecture',whichcanbeusedtoseparatetheuser applications and the physical database.

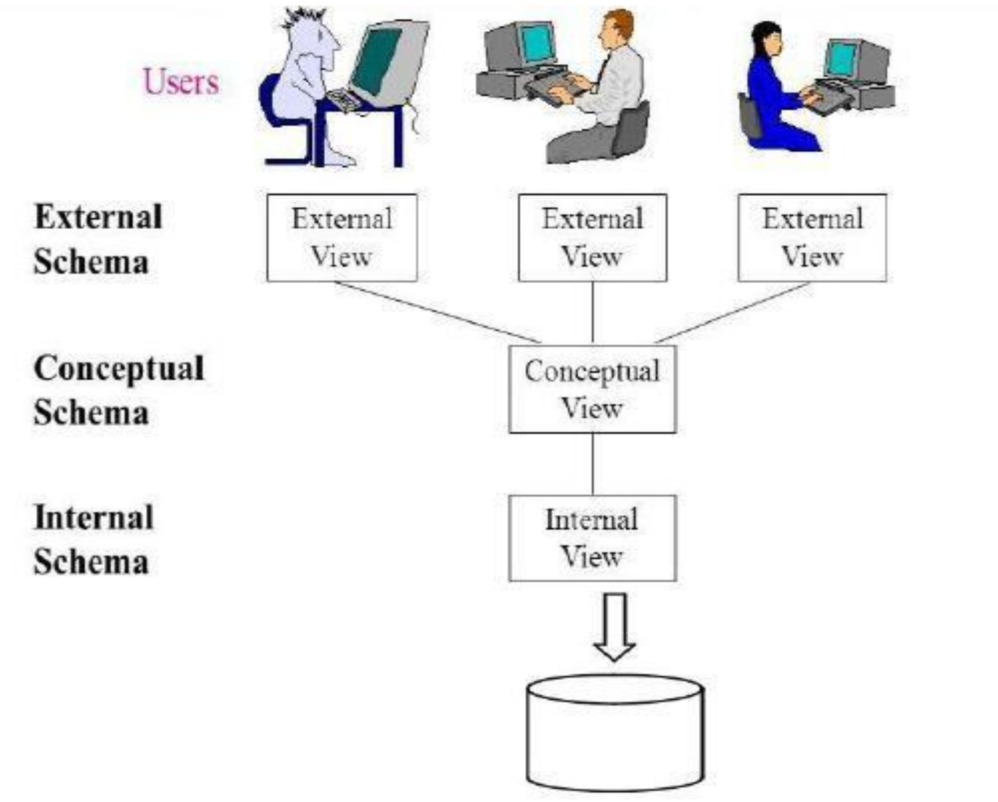


Figure:LevelsofAbstractioninaDBMS

1. PhysicalLevel:(orInternalView/Schema):

The lowest level of abstraction describes *how* the data are actually stored. The physical level describes complex low-level data structures in detail..

Example:

Customeraccountdatabasecanbedescribed.

2. LogicalLevel:(orConceptualView/Schema):

Thenext-higherlevelof abstractiondescribes*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**.

Example:

Each record

typecustomer=record

cust_name: sting;

cust_city: string;

cust_street: string;end;

3. ConceptualLevel:(orExternalView/ Schema):

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 ofinformation stored in a largedatabase. 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 simplifytheirinteraction with thesystem. Thesystem mayprovidemany views for the same database.

Example:

Forexample,wemay describearecordas follows:

type*instructor* =**record**

ID: **char**(5);

name: **char**(20);

dept name : **char** (20);

salary:**numeric**(8,2);

end;

Thiscodedefinesanewrecordtypecalled*instructor*withfourfields.Eachfieldhasaname and a type associated with it. A university organization may have several such record types, including

• *department*,withfields*dept_name*,*building*,and*budget*
• *course,*withfields *course_id*,*title*, *dept_name*,and *credits*
• *student*,withfields*ID*,*name*,*dept_name*,and*tot_cred*

At the physical level, an *instructor, department,* or *student* record can be described as a block of consecutive storage locations. At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Finally, at the view level, computer users see a set of application programs that hide details of the data types. At the view level, several views of the database aredefined, and a database user sees some or all of these views.

UNIT II

DATABASEDESIGN

ERModel

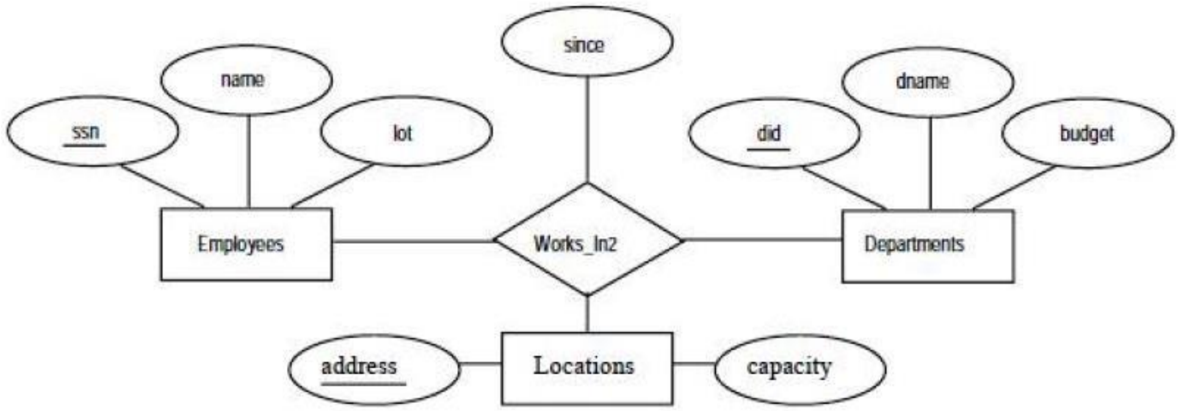
Data Models

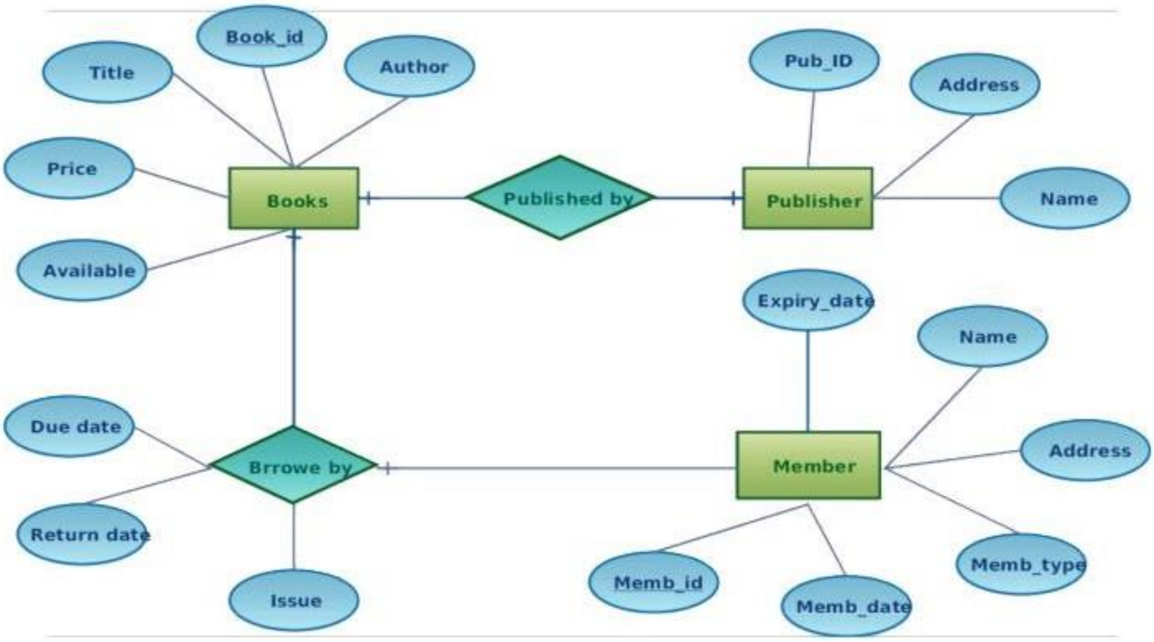
Underlyingthestructureofadatabaseisthe**datamodel**:acollectionofconceptualtoolsfor describing data, data relationships, data semantics, and consistency constraints.

Thedata modelscan be classified intofour different categories:

- **Relational Model**. The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as **relations**. The relational model is an example of a record-based model.
- **Entity-Relationship Model**. The entity-relationship (E-R) data model uses a collection of basic objects, called *entities*, and *relationships* among these objects. Suppose that each department has offices in several locations and we want to record the locations at which each employee works. The ER diagram for this variant of Works In, which we call Works In2

Example-ternary





E-R Diagram for Library Management System

ERModel -(RailwayBookingSystem)

ERModel-(BankingTransactionSystem)

Object-Based Data Model.

Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object oriented data model that can be seen as extending the E-R model with notions of encapsulation methods (functions), and object identity.

Semi-structuredDataModel.

The semi-structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The **Extensible Markup Language (XML)** is widely used to represent semi structured data.

Historically, the **network data model** and the **hierarchical data model** preceded the relational data model.

These models were tied closely to the underlying implementation, and complicated the taskof modelling data.

As a result they are used little now, except in old database code that is still in service in some places.

DatabaseLanguages

Adatabasesystem provides a **data-definition language** to specifythedatabase schemaand 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-ManipulationLanguage

A**data-manipulationlanguage(DML)**isalanguagethatenablesuserstoaccessor manipulate data as organized by the appropriate data model. The types of access are:

• Retrievalofinformationstoredinthedatabase

• Insertionofnewinformationintothe database

• Deletionofinformationfromthedatabase

• Modificationofinformationstoredinthedatabase

There are basically two types:

• **ProceduralDMLs**requireauserto specify*what*dataareneeded and *how* to get thosedata.

• **DeclarativeDMLs**(alsoreferredtoas**nonproceduralDMLs**)requireausertospecify *what*dataareneeded*without*specifying howto get thosedata.

A **query** is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a **query language**. Although technically incorrect, itis common practice to use the terms *query language* and *data-manipulation language* synonymously.

Data-DefinitionLanguage(DDL)

Wespecifyadatabaseschemaby aset ofdefinitions expressedby aspecial language called a **data definition language** (**DDL**). The DDL is also used to specify additional properties of the data.

• **Domain Constraints**. A domain of possible values must be associated with every attribute (for example, integertypes, charactertypes, date/timetypes). Declaring an attributeto beofa particular domain acts as a constraint on the values that it can take. Domain constraints arethe most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.

• **Referential Integrity**. There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation(referentialintegrity).Forexample,thedepartmentlisted foreachcoursemustbeone thatactually exists.Moreprecisely,the*deptname* valuein a*course*recordmustappearinthe *dept name* attribute of some record of the *department* relation.

• **Assertions**. An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions. However,there are many constraints that we cannot express by using only these special forms. For example, “Every department must have at least five courses offered every semester” must be expressed as an assertion..

• **Authorization**. We may want to differentiate among the users as far as the type of access theyarepermittedonvariousdatavaluesinthedatabase.Thesedifferentiationsareexpressed in terms of **authorization**, the most common being: **read authorization**, which allows reading, but not modification, of data; **insert authorization**, which allows insertion of new data, but not modification of existing data; **update authorization**, which allowsmodification, but not deletion, of data; and **delete authorization**, which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization. The DDL, just like any other programming language, gets as input some instructions (statements) and generates some output. The output of the DDL is placed in the **data dictionary**,which contains **metadata**—that is, data about data.

Data Dictionary

We can define a data dictionary as a DBMS component that stores the definition of data characteristics and relationships. You may recall that such “data about data” were labelled metadata. The DBMS data dictionary provides the DBMS with its self describing characteristic. In effect, the data dictionary resembles and X-ray of the company’s entire data set, and is a crucial element in the data administration function.

Forexample,thedatadictionarytypicallystoresdescriptionsof all:

• Data elements that are define in all tables of all databases. Specifically the data dictionary stores the name, data types, display formats, internal storage formats, and validation rules. The data dictionary tells where an element is used, by whom it is used and so on.

• Tablesdefineinalldatabases.Forexample,the datadictionaryislikelytostore thenameof the table creator, the date of creation access authorizations, the number of columns, and soon.

• Indexes define for each database tables. For each index the DBMS stores at least the index name the attributes used, the location, specific index characteristics and the creation date.

• Definedatabases:whocreatedeachdatabase,thedateofcreationwherethedatabaseis located, who the DBA is and so on.

• EndusersandTheAdministratorsofthedatabase

• Programsthataccessthedatabaseincludingscreenformats,reportformatsapplication formats, SQL queries and so on.

• Access authorizationforallusersofalldatabases.

• Relationships among data elements which elements are involved: whether the relationship are mandatory or optional, the connectivity and cardinality and so on.

DatabaseAdministratorsandDatabaseUsers

A primary goal of a database system is to retrieve information from and store new information in the database.

DatabaseUsersandUser Interfaces

There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

Naive users are unsophisticated users who interact with the system by invoking one of the applicationprogramsthathavebeenwrittenpreviously.Forexample,abanktellerwhoneeds to transfer $50 from account *A* to account *B* invokes a program called *transfer*.

Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. **Rapid application development (RAD)** tools are tools that enable an application programmer to construct forms and reports without writing a program.

Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **query processor**, whosefunction is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the databasefall in this category.

Online analytical processing (OLAP) tools simplify analysts' tasks by letting them view summariesofdataindifferentways. Forinstance, ananalyst canseetotalsalesbyregion(for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region).

DatabaseArchitecture:

The architecture of a database system is greatly influenced by the underlying computersystem 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.

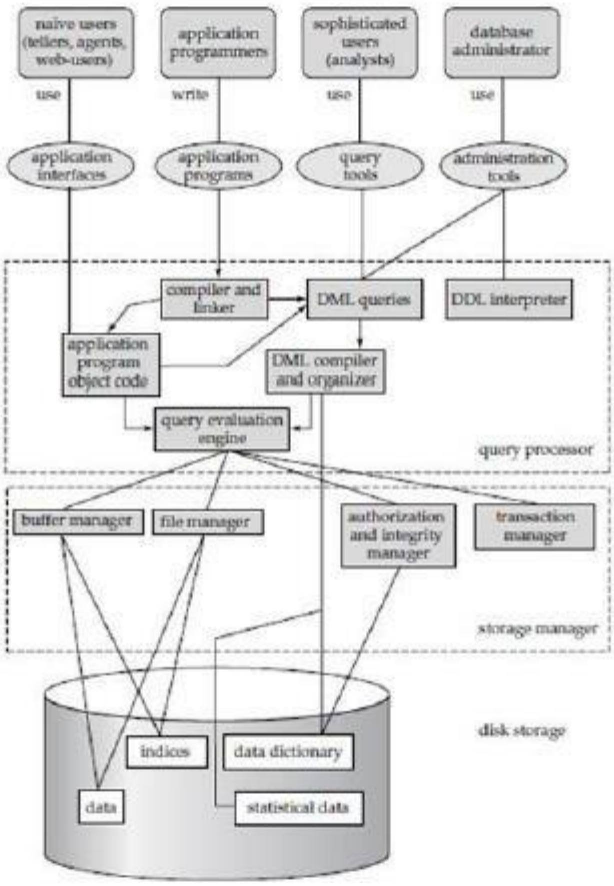


Figure:DatabaseSystem Architecture

Adatabasesystemispartitionedinto modulesthatdealwitheachoftheresponsibilitiesofthe 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. The query processor is important because it helps the database system simplify and facilitate access to data.

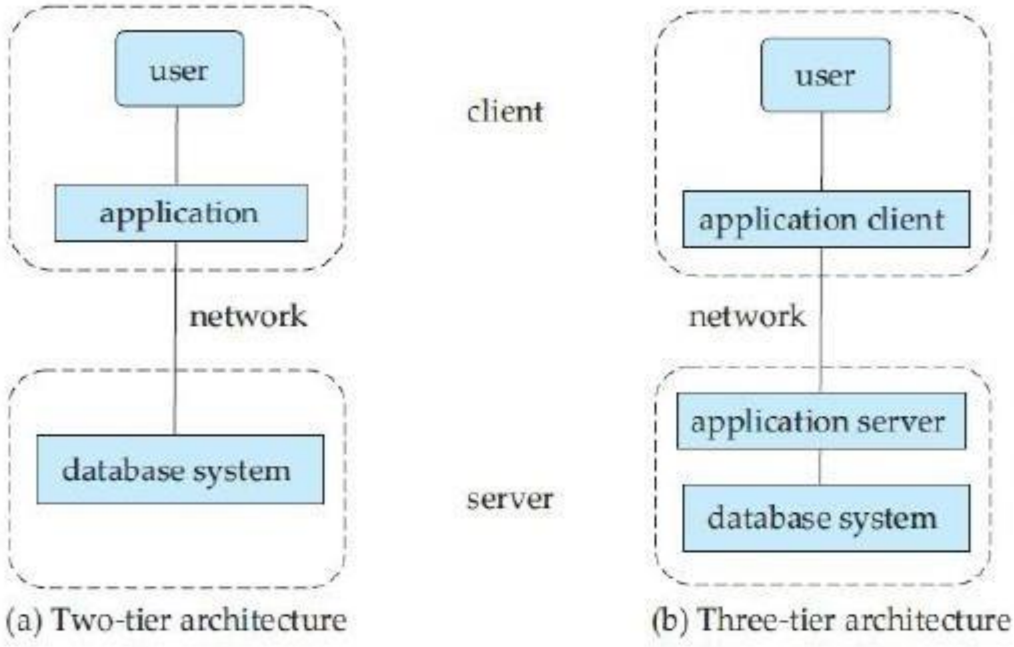


Figure:Two-tierandthree-tier architectures.

QueryProcessor:

Thequeryprocessor components include

· **DDLinterpreter,** whichinterpretsDDLstatementsandrecordsthedefinitionsinthedata 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.

StorageManager:

A *storage manager* is a program module 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 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.

Transaction Manager:

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.

ConceptualDatabaseDesign-EntityRelationship(ER)Modeling:

Database Design Techniques

1. ERModeling (Topdown Approach)
2. Normalization(BottomUpapproach)

What is ER Modeling?

Agraphicaltechniqueforunderstandingandorganizingthedataindependentoftheactual database implementation

Weneedtobe familiar withthefollowing termstogo further.

Entity

Anythingthathasanindependentexistenceandaboutwhichwecollectdata.Itisalso known as entity type. In ER modeling, notation for entity is given below.

Entityinstance

Entityinstanceisaparticularmemberoftheentitytype. Example

for entity instance : A particular employee

Regular Entity

Anentitywhichhasitsownkeyattributeisaregularentity.

Example for regular entity : Employee.

Weakentity

Anentitywhichdependsonotherentityforitsexistenceanddoesn'thaveanykeyattributeof its own is a weak entity.

Example for a weak entity : In a parent/child relationship, a parent is considered as a strongentity and the child is a weak entity.

InERmodeling,notationforweakentityisgiven below.

Attributes

Properties/characteristicswhichdescribeentitiesarecalledattributes. In ER modeling, notation for attribute is given below

- **DomainofAttributes**

  The set of possible values that an attribute can take is called the domain of the attribute.

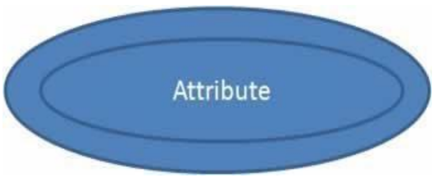
  For example, the attribute day may take any value from the set {Monday, Tuesday ... Friday}. Hence this set canbe termed as the domain of the attribute day.

- **Key attribute**

  The attribute (or combination of attributes) which is unique for every entity instanceis called key attribute.

  E.g the employee_id of an employee, pan_card_number of a person etc.If the key attributeconsistsoftwoormoreattributesincombination,itiscalledacompositekey. In ER modeling, notation for key attribute is given below.

- **Simple attribute**
  Ifanattributecannotbedividedintosimplercomponents,itisasimpleattribute. Example for simple attribute : employee_id of an employee.
- **Compositeattribute**
  Ifanattribute canbesplitintocomponents,it iscalledacompositeattribute.
  Exampleforcompositeattribute:Nameoftheemployeewhichcanbesplitinto First_name, Middle_name, and Last_name.
- **Single valued Attributes**
  Ifan attributecan takeonlyasinglevalueforeachentityinstance, itisasinglevalued attribute.
  example for single valued attribute : age of a student. It can take only one value for a particular student.
- **Multi-valued Attributes**
  Ifanattributecantakemorethanonevalueforeachentityinstance,itisamulti- valued attribute.
  exampleformultivaluedattribute:telephonenumberofanemployee,aparticular employee may have multiple telephone numbers.
  InERmodeling,notationformulti-valuedattributeisgivenbelow.



- **StoredAttribute**
  Anattributewhichneedtobestoredpermanentlyisastoredattribute Example for stored attribute : name of a student
- **DerivedAttribute**
  An attribute which can be calculated or derived based on other attributes is a derived attribute.

Example for derived attribute : age of employee which can be calculated from date of birth and current date.

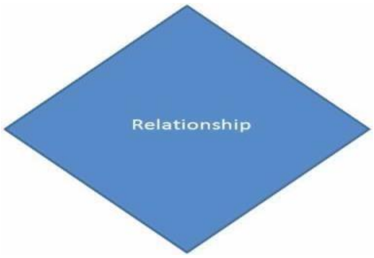
InERmodelling,notationforderivedattributeisgiven below.

Relationships

Associationsbetween entitiesarecalledrelationships

Example:Anemployeeworksforanorganization.Here"worksfor"isarelationbetweenthe entities employee and organization.

InERmodeling,notation forrelationshipisgivenbelow.



HoweverinERModeling,ToconnectaweakEntitywithothers,youshoulduseaweak relationship notation as given below

DegreeofaRelationship

Degree of a relationship is the number of entity types involved. The n-ary relationship is the general form for degree n. Special cases are unary, binary, and ternary ,where the degree is 1, 2, and 3, respectively.

Exampleforunaryrelationship:Anemployeeiaamanagerofanother employee Example for binary relationship : An employee works-for department.

Exampleforternaryrelationship:customerpurchaseitem fromashopkeeper **Cardinalityof a Relationship** Relationship cardinalities specify how many of each entity type is allowed.

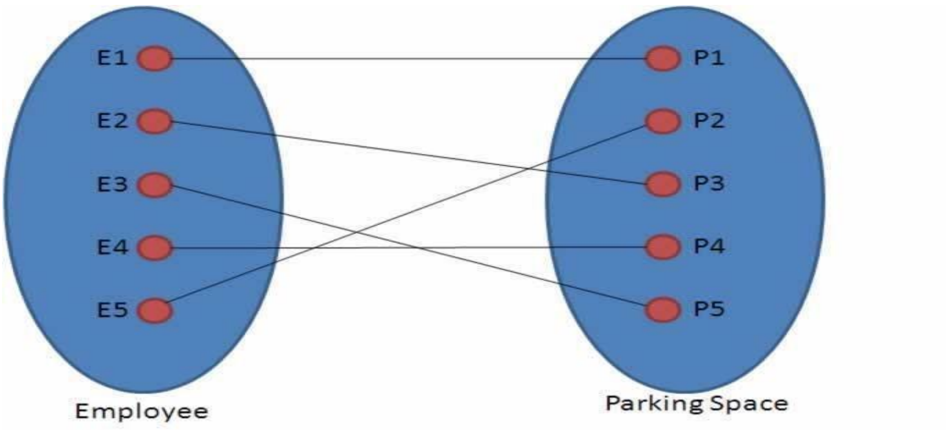
Relationshipscanhavefourpossibleconnectivitiesasgivenbelow.

1. Oneto one(1:1)relationship
2. Oneto many(1:N)relationship
3. Manytoone(M:1) relationship
4. Manyto many(M:N) relationship

Theminimumandmaximumvaluesofthisconnectivityiscalledthecardinalityofthe relationship

ExampleforCardinality –One-to-One(1:1)

Employeeisassigned withaparking space.

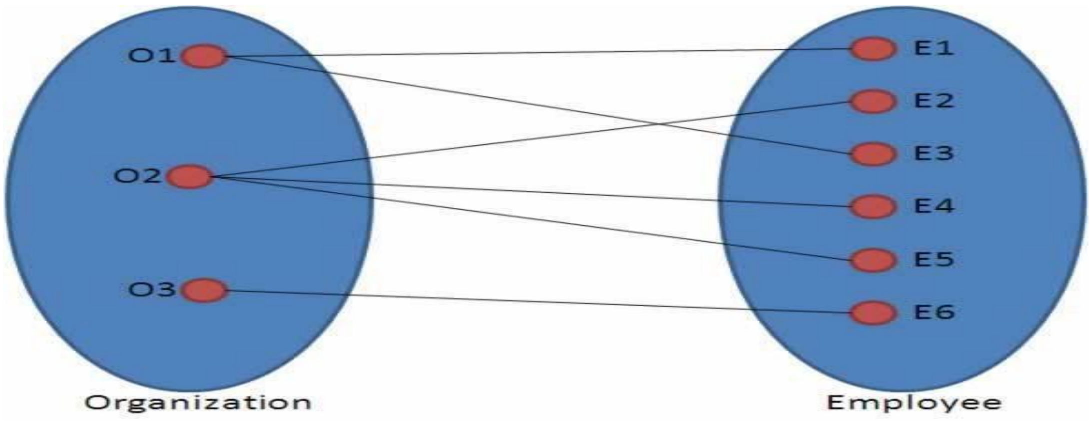


Oneemployeeisassignedwithonlyoneparkingspaceandoneparkingspaceisassignedto only one employee. Hence it is a 1:1 relationship and cardinality is One-To-One (1:1)

InERmodeling,thiscanbementionedusingnotationsasgiven below

ExampleforCardinality –One-to-Many(1:N)

Organizationhasemployees

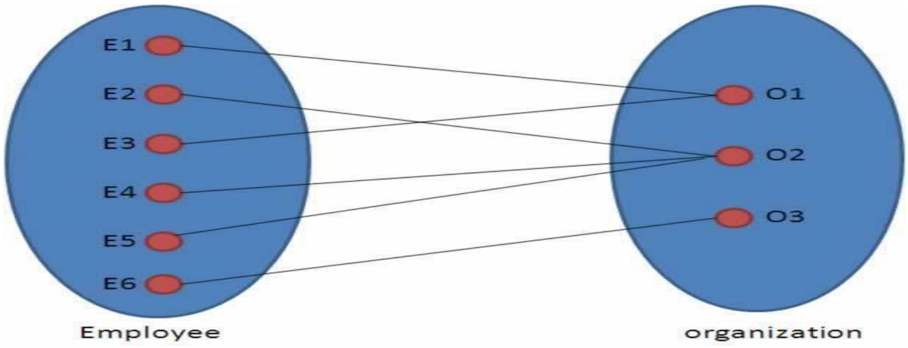


Oneorganizationcanhavemanyemployees,butoneemployeeworksinonlyone organization. Hence it is a 1:N relationship and cardinality is One-To-Many

(1:N)InERmodeling,thiscanbementionedusingnotationsasgivenbelow

ExampleforCardinality –Many-to-One(M:1)

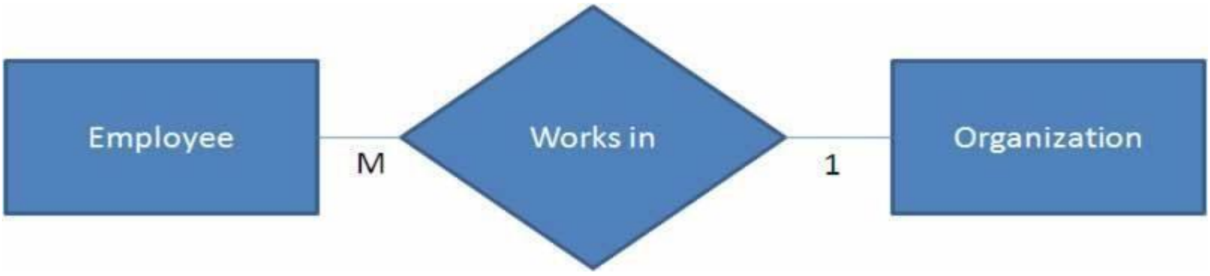
Itis the reverseof theOneto Many relationship.employeeworks in organization



Oneemployeeworksinonlyoneorganization Butoneorganizationcanhavemany employees.

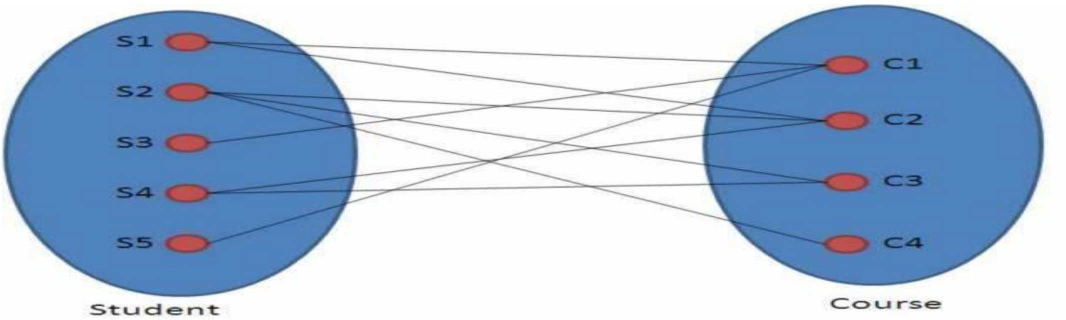
Hence it is a M:1 relationship and cardinality is Many-to-One (M

:1)InERmodeling,thiscanbementionedusingnotationsasgivenbelow.



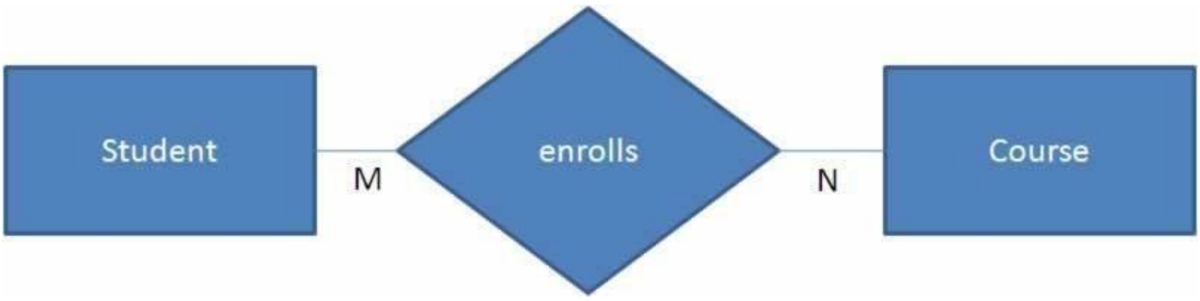
Cardinality–Many-to-Many (M:N)

Studentsenrolls forcourses



One student canenroll for many coursesand one course can be enrolledby many students.Hence it is a M:N relationship and cardinality is Many-to-Many (M:N)

InERmodeling,this canbementionedusingnotationsasgiven below



RelationshipParticipation

1. Total

Intotalparticipation,everyentityinstancewillbeconnectedthroughtherelationshipto another instance of the other participating entity types

2. Partial

Example for relationship participation Consider the relationship - Employee is head of the department. Here all employees will not be the head of the department. Only one employee will be the head of the department. In other words, only few instances of employee entity participate in the above relationship. So employee entity's participation is partial in the said relationship.

AdvantagesandDisadvantagesofERModeling(MeritsandDemeritsofERModeling) Advantages

1. ERModelingissimpleandeasilyunderstandable.Itisrepresentedinbusinessusers language and it can be understood by non-technical specialist.
2. IntuitiveandhelpsinPhysicalDatabase creation.
3. Canbegeneralizedandspecializedbasedonneeds.
4. Canhelpindatabase design.
5. Givesahigherlevel description of the system.

Disadvantages

1. PhysicaldesignderivedfromE-RModelmayhavesomeamountofambiguitiesorinconsistency.
2. Sometimediagramsmayleadtomisinterpretations

Relational Model

The relational model is today the primary data model for commercial data processing applications. It attained its primary position because of its simplicity, which eases the job of the programmer,compared to earlier data models such as the network model or the hierarchical model.

StructureofRelational Databases:

A relational database consists of a collection of **tables**, each of which is assigned a unique name.

For example, consider the *instructor* table of Figure:1.5, which stores information about instructors.Thetablehas fourcolumnheaders: *ID*,*name*,*deptname*,and*salary*.Eachrowof this table records information about an instructor, consisting of the instructor's *ID*, *name*,*dept name*, and *salary*.

DatabaseSchema

When we talk about a database, we must differentiate between the **database schema**, whichis the logical design of the database, and the **database instance**, which is a snapshot of the data in the database at a given instant in time. The concept of a relation corresponds to the programming language notion of a variable, while the concept of a **relation schema** corresponds to the programming-language notion of type definition.

Keys

Akeyconstraintisastatementthatacertainminimalsubsetofthefieldsofarelationisaunique identifier for a tuple.

Example:

The‘students’relationandtheconstraintthatno2studentshavethasamestudentid(sid). These can be classified into 3 types as below.

Primary Key:

This is also a candidate key, whose values are used to identify tuples in the relation.It is common to designate one of the candidate keys as a primary key of the relation.The attributes that form the primary key ofarelationschemaareunderlined.It isusedtodenoteacandidatekeythat ischosenby the database designer as theprincipal means of identifying entities with an entity set.

A**superkey:**

is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation. For example, the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a superkey. The *name* attribute of *instructor*, on the other hand, is not a superkey, because several instructors might have the same name. A superkey may contain extraneous attributes. For example, the combination of *ID* and *name* is a superkey for the relation *instructor*. If *K* is a superkey, then so is any supersetof*K*.Weareofteninterestedinsuperkeysforwhichnopropersubsetisasuperkey. Suchminimalsuperkeysarecalled

candidate keys:

Itiscustomarytolisttheprimarykeyattributesofarelationschemabeforetheotherattributes;

for example, the *dept name* attribute of *department* is listed first, since it is the primary key. Primary key attributes are also underlined. A relation, say *r*1, may include among itsattributes the primary key of another relation, say *r*2. This attribute is called a **foreign key** from *r*1, referencing *r*2.

SchemaDiagrams

A database schema, along with primary key and foreign key dependencies, can be depictedby **schema diagrams**.

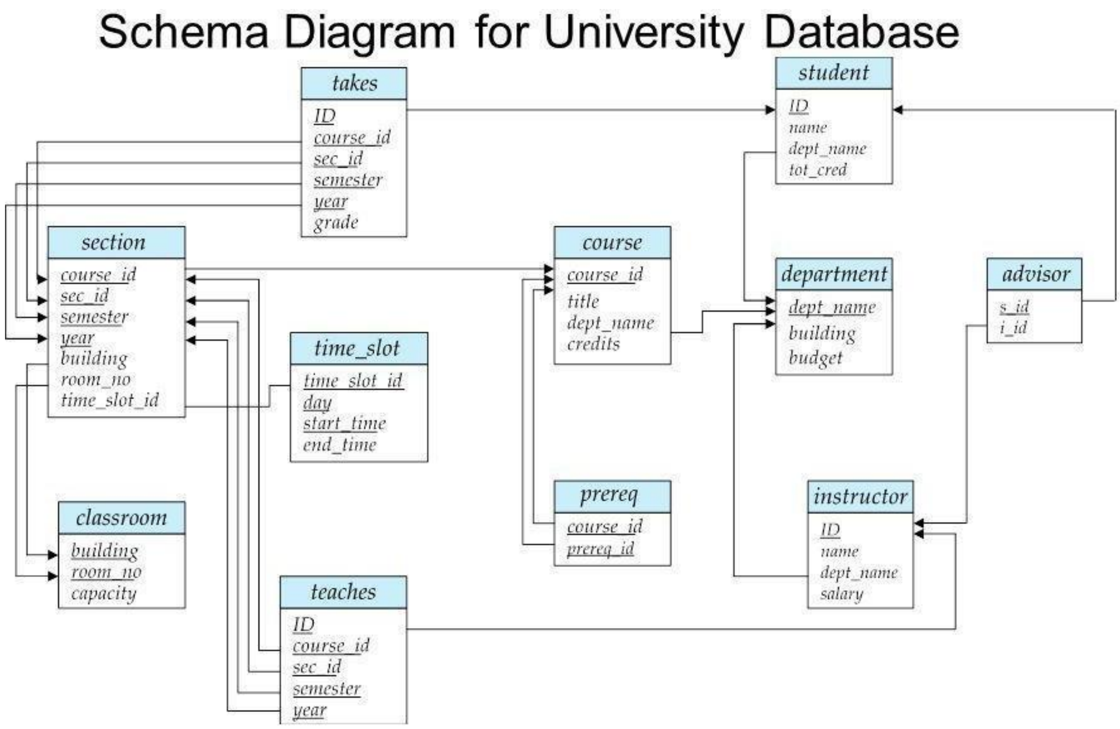


Figure2.5:Schemadiagram fortheuniversitydatabase.

Referential integrity constraints other than foreign key constraints are not shown explicitly in schema diagrams. We will study a different diagrammatic representation called the entity-relationship diagram.

UNIT III

STRUCTUREDQUERYLANGUAGE

What is SQL?

- SQLisStructuredQueryLanguage,whichisadatabaselanguagedesignedforthe retrieval and management of data in a relational database.
- AlltheRDBMSsystemslikeMySQL,MSAccess,Oracle,Sybase,Postgres,and SQL Server use SQL as their standard database language.

WhytoUseSQL?

SQLprovidesaninterfacetoarelationaldatabase.Here

, are important reasons for using SQL

- Ithelps usersto accessdata inthe RDBMSsystem.
- Ithelpsus todescribethedata.
- Itallowsus todefinethedataina databaseandmanipulate thatspecificdata.
- WiththehelpofSQLcommandsinDBMS,wecancreateanddropdatabasesand tables.
- SQLoffersusto usethe functionin adatabase,createaview,andstored procedure.
- Wecansetpermissionsontables,procedures,andviews.

HistoryofSQL

"ARelationalModelofDataforLargeSharedDataBanks"wasapaper whichwaspublished by the great computer scientist "E.F. Codd" in 1970.

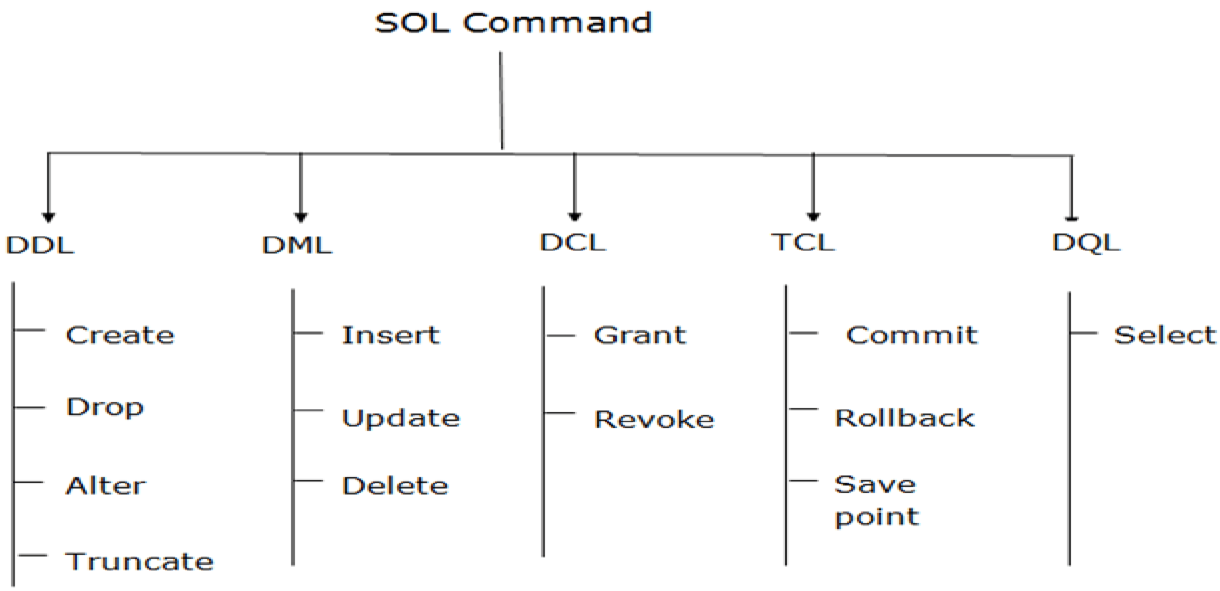
The IBM researchers Raymond Boyce and Donald Chamberlin originally developed the SEQUEL (Structured English Query Language) after learning from the paper given by E.F. Codd.TheybothdevelopedtheSQLattheSanJoseResearchlaboratoryof IBMCorporation in 1970. In 1979, Relational Software, Inc. (now Oracle) introduced the first commercially available implementation of SQL.

SQL became astandardof theAmerican National Standards Institute(ANSI) in 1986, and of theInternational Organization for Standardization(ISO) in 1987.[11]Since then, the standard has been revised to include a larger set of features. Despite the existence of standards, most SQL code requires at least some changes before being ported to different databasesystems. New versions of the standard were published and most recently, 2016.

Typesof SQL

Hereare fivetypes of widely usedSQL queries.

- DataDefinitionLanguage(DDL)
- DataManipulationLanguage(DML)
- DataControlLanguage(DCL)
- TransactionControl Language(TCL)
- DataQueryLanguage(DQL)



All operations performed on the information in a database are run using SQL **statements**. ASQLstatementconsistsofidentifiers,parameters,variables,names,datatypes,andSQL **reserved words**.

Whatis DDL?

Definition: The Language used to define thedatabase structure or schema is called"Data Definition Language".

- TheCommands(or) statementsusedto definethestructuteofdatabaseare:

1. CREATE
2. ALTER
3. DROP
4. TRUNCATE
5. RENAME

1. CREATE

Createcommandcanbe usedtocreate

(i) Databases

(ii) Tablesand

(iii) Views.

(i) CreatingDatabase

Syntax:

```
createdatabasedabasename;
```

Ex:createdatabaseMRCET_ITA;

(ii) CreatingTable

Syntax:

```
Createtabletablename(Columnname1Datatype,
        Columnname2   Datatype,

                    ……..,

Columnnamendatatype);
```

Ex:createtableStudent(SRno integer(5),

Snamevarchar(20),

Addressvarchar(15));

2. ALTER Command

- TheALTERTABLEstatementisusedtoadd,delete,ormodifycolumnsinan existing table.
- TheALTERTABLEstatementisalsousedtoadd anddropvariousconstraintsonan existing table.

1. ALTERTABLE -ADDColumn

Toadd acolumn ina table,usethefollowing syntax:

```
ALTERTABLEtable_name
ADDcolumn_namedatatype;
```

Ex: ThefollowingSQLaddsan"Email"columntothe "Customers" table:

```
ALTERTABLECustomers
ADDEmailvarchar(255);
```

ALTER TABLE - DROP COLUMN

Todeleteacolumninatable,usethefollowingsyntax(noticethatsomedatabasesystems don't allow deleting a column):

Syntax:

```
ALTERTABLEtable_name
DROPCOLUMNcolumn_name;
```

Ex: Thefollowing SQLdeletes the "Email" columnfrom the"Customers"table:

ALTERTABLE Customers
DROPCOLUMNEmail;

ALTERTABLE-ALTER/MODIFYCOLUMN

Tochangethedata type of acolumnin atable, usethe following syntax:

SQLServer/MS Access:

```
ALTERTABLEtable_name
ALTERCOLUMNcolumn_namedatatype;
```

Ex:ALTER TABLE supplier

```
ALTERCOLUMNsupplier_nameVARCHAR(100)NOTNULL;
```

MySQL/Oracle(prior version 10G):

```
ALTERTABLEtable_name
MODIFYCOLUMNcolumn_namedatatype;
```

Example1:Modifyingsingle Column

```
ALTERTABLE supplier
MODIFYsupplier_namechar(100)NOT NULL;
```

Example2:ModifyingMultiple Columns

```
ALTERTABLE supplier
MODIFYsupplier_nameVARCHAR(100)NOTNULL, MODIFY
city VARCHAR(75);
```

Oracle10Gand later:

```
ALTERTABLEtable_name
MODIFYcolumn_namedatatype;
```

3.Drop Command

Syntax

Todrop acolumnin an existingtable, theSQLALTER TABLE syntax is:

```
ALTER TABLE table_name
  DROPCOLUMNcolumn_name;
```

Example

Let'slookatanexamplethatdrops(ie:deletes)acolumnfromatable. For

example:

```
ALTERTABLE supplier
  DROP COLUMNsupplier_name;
```

ThisSQLALTERTABLEexamplewilldropthecolumn called ***supplier_name***from the table called ***supplier***.

TRUNCATE:

Thiscommandusedtodeletealltherowsfromthetableandfreethespacecontainingthe table.

Syntax:

```
TRUNCATETABLE table_name;
```

Example:

```
TRUNCATEtablestudents;
```

WhatisDataManipulationLanguage?

Data Manipulation Language (DML) allows user to modify the database instance byinserting, modifying, and deleting its data. It is responsible for performing all typesdata modification in a database.

There are three basic constructs which allow database program and user to enter data and information are:

Herearesomeimportant DMLcommandsin SQL:

- INSERT
- UPDATE
- DELETE

INSERT: This statement is a SQL query. This command is used to insert data into the row of a table.

Syntax:

```
INSERTINTOTABLE_NAME(col1,col2,col3,. ........ col N)
VALUES(value1,value2,value3,....... valueN);
Or
INSERTINTO TABLE_NAME
VALUES(value1,value2, value3,...... valueN);
```

For example:

```
INSERTINTOstudents(RollNo,FIrstName,LastName)VALUES('60','Tom', Erichsen');
```

UPDATE:

Thiscommand is used to update ormodify thevalueof acolumn in thetable.

Syntax:

```
UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]
```

For example:

```
UPDATEstudents
SETFirstName='Jhon',LastName='Wick' WHERE
StudID = 3;
```

DELETE:

Thiscommand is used to remove oneor morerows from a table.

Syntax:

```
DELETEFROMtable_name[WHEREcondition];
```

For example:

```
DELETE FROM students
WHEREFirstName='Jhon';
```

Whatis DCL?

DCL (Data Control Language) includes commands like GRANT and REVOKE, which are useful to give "rights & permissions." Other permission controls parameters of the database system.

ExamplesofDCL commands:

Commandsthatcomeunder DCL:

- Grant
- Revoke

Grant:

Thiscommand isuse togive useraccess privilegesto a database.

Syntax:

```
GRANTSELECT,UPDATEON MY_TABLETOSOME_USER, ANOTHER_USER;
```

For example:

```
GRANTSELECT ONUsers TO'Tom'@'localhost;
```

Revoke:

Itisusefultobackpermissionsfromtheuser.

Syntax:

```
REVOKEprivilege_nameONobject_nameFROM{user_name|PUBLIC |role_name}
```

For example:

```
REVOKESELECT,UPDATEONstudentFROMBCA,MCA;
```

What is TCL?

TransactioncontrollanguageorTCLcommandsdealwiththetransactionwithinthe database.

Commit: This commandisused to saveall the transactionsto the database.

Syntax:

```
Commit;
```

For example:

```
DELETEFROMStudents
WHERE RollNo =25;
COMMIT;
```

Rollback

Rollback command allows you to undo transactions that have not already been saved to thedatabase.

Syntax:

```
ROLLBACK;
```

Example:

```
DELETEFROMStudents
WHERE RollNo =25;
```

SAVEPOINT

Thiscommand helpsyou tosets asavepoint withinatransaction.

Syntax:

```
SAVEPOINTSAVEPOINT_NAME;
```

Example:

```
SAVEPOINTRollNo;
```

What is DQL?

DataQueryLanguage(DQL)isusedtofetchthedatafromthedatabase.Itusesonlyone command:

SELECT:

Thiscommandhelpsyoutoselecttheattributebasedontheconditiondescribedbythe WHERE clause.

Syntax:

```
SELECTexpressions
FROM TABLES
WHERE conditions;
```

For example:

```
SELECTFirstName
FROM Student
```

```
WHERERollNo>15;
```

TCL Commands

TCL Commands in SQL- Transaction Control Language Examples: Transaction Control Language can be defined as the portion of a database language used for maintaining consistency of the database and managing transactions in database. A set of SQL statements that are co-related logically and executed on the data stored in the table is known as transaction. In this tutorial, you will learn different TCL Commands in SQL with examples and differences between them.

1. CommitCommand
2. Rollback Command
3. SavepointCommand

TCLCommandsinSQL-Transaction ControlLanguage Examples

The modifications made by the DML commands are managed by using TCL commands. Additionally, it makes the statements to grouped together into logical transactions.

TCL Commands

Therearethree commands thatcomeundertheTCL:

1. **Commit**

The main use of Commit command is to make the transaction permanent. If there is a needfor any transaction to be done in the database that transaction permanent through commit command.

Syntax:

COMMIT;

ForExample

UPDATESTUDENTSETSTUDENT_NAME=‘Maria’WHERESTUDENT_NAME=
‘Meena’;
COMMIT;

- By using the above set of instructions, you can update the wrong student name by the correct one and save it permanently in the database. The update transaction gets completed when commit is used. If commit is not used, then there will be lock on ‘Meena’ record till the rollback or commit is issued.

- Now have a look at the below diagram where ‘Meena’ is updated and there is a lock on her record. The updated value is permanently saved in the database after the use of commit and lock is released.

2. **Rollback**

- Usingthiscommand, thedatabase canbe restoredto thelastcommitted state.
- Additionally, it is also used with savepoint command for jumping to a savepoint in a transaction.

Syntax:

Rollbacktosavepoint-name;

<u>For example</u>

UPDATESTUDENTSETSTUDENT_NAME=‘Manish’WHERESTUDENT_NAME =‘Meena’;ROLLBACK;

- Thiscommandisusedwhentheuserrealizesthathe/shehasupdatedthewrong information after the student name and wants to undo this update.

- TheuserscanissuesROLLBACK commandandthenundo theupdate.

Havealookat thebelowtablesto knowbetterabout theimplementation ofthiscommand.

3. **Savepoint**

The main use of the Savepoint command is to save a transaction temporarily. This way users can rollback to the point whenever it is needed.

Thegeneralsyntax forthesavepointcommandismentioned below:

savepointsavepoint-name;

<u>ForExample</u>

Followingis thetableofaschool class

UsesomeSQL queriesontheabovetableandthen watch theresults

INSERTintoCLASS VALUES(101, ‘Rahul);

Commit;

UPDATECLASSSET NAME=‘Tyler’whereid=101;

SAVEPOINTA;

INSERTINTOCLASSVALUES(102, ‘Zack’);

SavepointB;

INSERTINTOCLASSVALUES(103, ‘Bruno’)

SavepointC;

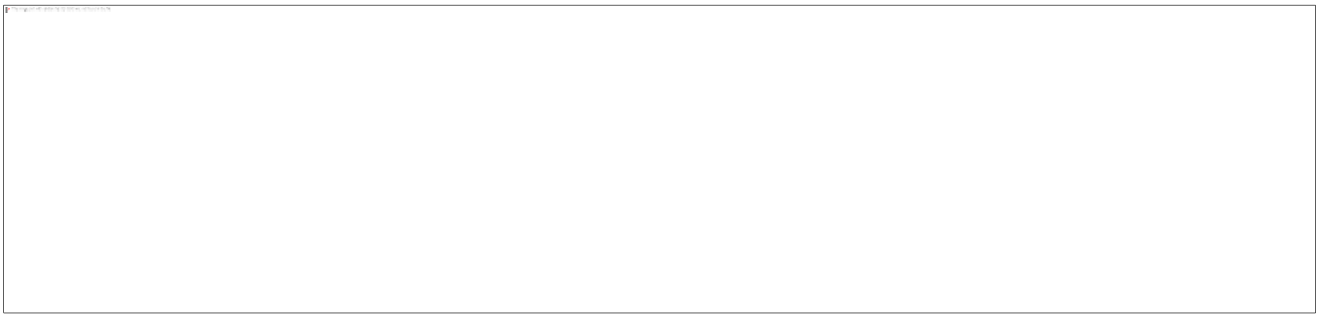
Select * from Class;

Theresultwilllooklike

NowrollbacktosavepointB

Rollback to B;

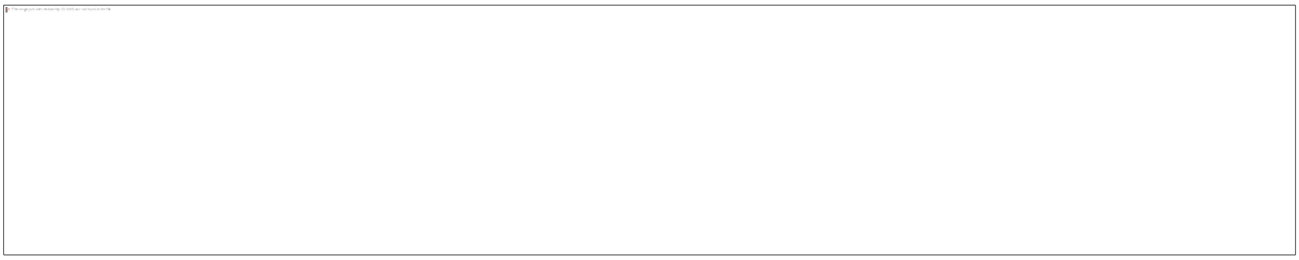
SELECT*from Class;



NowrollbacktosavepointA

rollback to A;

SELECT*from class;



Differencebetweenrollback, commit andsavepointtcl commandsinSQL.

	Rollback	Commit	Savepoint

1.	Databasecanberestoredtothelast committed state	Savesmodification made by DML Commands and it permanently saves the transaction.	Itsavesthetransaction temporarily.
2.	Syntax-ROLLBACK[To SAVEPOINT_NAME];	Syntax-COMMIT;	Syntax-SAVEPOINT [savepoint_name;]
3.	Example-ROLLBACK Insert3;	Example- SQL>COMMIT ;	Example-SAVEPOINT table_create;

```
STARTTRANSACTION;
savepointa;
updatet1setn1=18wheren1=13; rollbackto
a;
```

In relational database the data is stored as well as retrieved in the form of relations (tables). Table 1shows the relationaldatabase withonly onerelationcalled**STUDENT**which stores **ROLL_NO**, **NAME**, **ADDRESS**, **PHONE** and **AGE** of students.

ROLL_NO	**NAME**	**ADDRESS**	**PHONE**	**AGE**
1	RAM	DELHI	9455123451	18
2	RAMESH	GURGAON	9652431543	18
3	SUJIT	ROHTAK	9156253131	20
4	SURESH	DELHI	9156768971	18

Thesearesomeimportantterminologiesthatareusedintermsofrelation.

Attribute: Attributesarethepropertiesthatdefinearelation.e.g.;**ROLL_NO,NAME**etc.

Tuple: Each row in the relation is known as tuple. The above relation contains 4 tuples, one of which is shown as:

1	RAM	DELHI	9455123451	18

Degree: Thenumberofattributesintherelationisknownasdegreeoftherelation.The STUDENT relation defined above has degree 5.

Cardinality:Thenumberoftuplesinarelationisknownascardinality.TheSTUDENT relation defined above has cardinality 4.

Column: Columnrepresentsthesetofvaluesforaparticularattribute.Thecolumn ROLL_NO isextracted from relation STUDENT.

ROLL_NO
1
2
3
4

SQLSet Operations

Setoperationsallowthe resultsofmultiplequeries tobecombinedintoa singleresult set.

The**Set Operators** combine a similar type of data from two or more SQL database tables. It mixes the result, which is extracted from two or more SQL queries, into a single result.

Set operators combine more than one select statement in a single query and return a specific result set.

SetoperatorsincludeUNION,INTERSECT,andEXCEPT.

UNION

In SQL the UNION clause combines the results of two SQL queries into a single table of all matching rows. The two queries must result in the same number of columns and compatible datatypesinordertounite.AnyduplicaterecordsareautomaticallyremovedunlessUNION ALL is used.

Syntaxof UNION:

SELECTcolumn1,column2...... ,columnNFROMtable_Name1[WHEREconditions]

UNION

SELECTcolumn1,column2...... ,columnN FROMtable_Name2[WHEREconditions];

Asimpleexamplewouldbeadatabasehavingtablessales2005andsales2006thathave identicalstructuresbutareseparatedbecauseofperformanceconsiderations.AUNION query could combine results from both tables.

NotethatUNIONALLdoesnotguaranteetheorderofrows.Rowsfromthesecondoperand may appear before, after, or mixed with rows from the first operand. In situations where a specific order is desired, ORDER BY must be used.

NotethatUNIONALLmaybemuch fasterthanplain UNION.

sales2005	
person	**amount**
Joe	1000
Alex	2000

Bob	5000

sales2006

person	amount
Joe	2000
Alex	2000
Zach	35000

Executingthisstatement:

SELECT*FROMsales2005UNIONSELECT* FROMsales2006;

yieldsthisresultset,thoughtheorderoftherowscanvarybecausenoORDERBYclause was supplied:

person	amount
Joe	1000
Alex	2000
Bob	5000

Joe	2000
Zach	35000

UNIONALLgivesdifferentresults,becauseitwillnoteliminateduplicates.Executingthis statement:

SELECT*FROMsales2005UNIONALLSELECT*FROM sales2006;

wouldgivetheseresults,againallowing variance forthelackof an ORDERBY statement:

person	amount
Joe	1000
Joe	2000
Alex	2000
Alex	2000
Bob	5000
Zach	35000

INTERSECT

The SQL INTERSECT operator takes the results of two queries and returns only rows that appearinbothresultsets.ForpurposesofduplicateremovaltheINTERSECToperatordoes not distinguish between NULLs.

The INTERSECT operator removes duplicate rows from the final result set. The INTERSECTALLoperatordoesnotremoveduplicaterowsfromthefinalresultset,butifa row appears X times in the first query and Y times in the second, it will appear min(X, Y) times in the result set.

ThedatatypeandthenumberofcolumnsmustbethesameforeachSELECTstatementused with the INTERSECT operator.

Syntaxof INTERSECT

SELECTcolumn1,column2...... ,columnNFROMtable_Name1[WHEREconditions]

INTERSECT

SELECTcolumn1,column2...... ,columnNFROMtable_Name2[WHEREconditions];

Let'sunderstandthebelowexamplewhichexplainshowtoexecuteINTERSECToperatorin Structured Query Language:

Inthisexample,weusedtwotables.BothtableshavefourcolumnsEmp_Id,Emp_Name, Emp_Salary, and Emp_City.

Employee_details1:

EmpId	EmpName	EmpSalary	EmpCity
201	Sanjay	25000	Delhi
202	Ajay	45000	Delhi
203	Saket	30000	Aligarh

Employee_details2:

EmpId	EmpName	EmpSalary	EmpCity
203	Saket	30000	Aligarh
204	Saurabh	40000	Delhi
205	Ram	30000	Kerala
201	Sanjay	25000	Delhi

Suppose,wewanttoseeacommonrecordoftheemployeefromboththetablesinasingle output. For this, we have to write the following query in SQL:

SELECTEmp_NameFROMEmployee_details1

INTERSECT

SELECTEmp_NameFROMEmployee_details2;

EmpId	EmpName	EmpSalary	EmpCity
201	Sanjay	25000	Delhi
203	Saket	30000	Aligarh

EXCEPT

TheSQLEXCEPToperatortakesthedistinctrowsofonequeryand returnstherowsthatdo not appear in asecond result set. For purposes of row elimination and duplicate removal, the EXCEPT operator does not distinguish between NULLs. The EXCEPT ALL operator does not remove duplicates, but if a row appears X times in the first query and Y times in the second, it will appear max(X - Y, 0) times in the result set.

Notably,theOracleplatformprovidesaMINUSoperatorwhichisfunctionallyequivalentto the SQL standard EXCEPT DISTINCT operator.

ThefollowingexampleEXCEPTqueryreturnsallrowsfromtheOrderstablewhere Quantity is between 1 and 49, and those with a Quantity between 76 and 100.

Wordedanotherway;thequeryreturnsallrowswheretheQuantityisbetween1and100, apart from rows where the quantity is between 50 and 75.

SELECT*FROMOrdersWHEREQuantityBETWEEN1AND100

EXCEPT

SELECT*FROMOrders WHEREQuantityBETWEEN50AND 75;

Joins

Ajoinisaquerythatcombinesrowsfromtwoormoretables,views, based on a common field between them.

Considerthefollowingtwotables−

Table1−CUSTOMERS Table

```
+---+-----------+----+--------+-------+
|ID|NAME        |AGE|ADDRESS      |SALARY|
+---+-----------+----+--------+-------+
|1|Ramesh|32|Ahmedabad|2000.00|
|2|Khilan|25|Delhi|1500.00|
|3|kaushik|23|Kota|2000.00|
|4|Chaitali|25|Mumbai|6500.00|
|5|Hardik|27|Bhopal|8500.00|
|6|Komal|22|MP         |4500.00|
|7|Muffy|24|Indore|10000.00|
+---+-----------+----+--------+-------+
```

Table2−ORDERSTable

```
+----+-------------+---------+-----+
|OID|DATE                  |CUSTOMER_ID|AMOUNT|
+----+-------------+---------+-----+
|102|2009-10-0800:00:00|3|3000|
```

```
|100|2009-10-0800:00:00|3|1500|
|101|2009-11-2000:00:00|2|1560|
|103|2008-05-2000:00:00|4|2060|
+- - - -+- - - - - - - - - - - - - -+- - - - - - - - -+- - - - - -+
```

Now,let usjoin thesetwotables inour SELECT statementas shown below.

```
SELECTID,NAME,AGE,AMOUNTFROMCUSTOMERS,ORDERS
WHERECUSTOMERS.ID= ORDERS.CUSTOMER_ID;
```

Thiswould producethe following result.

```
+- - -+-----------+- - - -+- - - - - -+
|ID |NAME       |AGE|AMOUNT|
+- - -+-----------+- - - -+- - - - - -+
|3|kaushik|23|3000|
|3|kaushik|23|1500|
|2|Khilan|25|1560|
|4|Chaitali|25|2060|
+- - -+-----------+- - - -+- - - - - -+
```

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; theycan all be used to join tables. However, the most common operator is the equal to symbol.

SQLJOINS:EQUIJOIN and NONEQUI JOIN

Thearetwotypes ofSQLJOINS-EQUI JOINandNON EQUI JOIN

1) SQL EQUI JOIN:

TheSQLEQUIJOINis asimpleSQLjoinusestheequalsign(=)asthecomparisonoperator for the condition. It has two types - SQL Outer join and SQL Inner join.

2) SQLNON EQUIJOIN :

TheSQLNON EQUIJOINis ajoin uses comparisonoperator other thantheequal sign like >,<, >=,<=withthecondition.

SQLEQUIJOIN:INNERJOIN andOUTERJOIN

TheSQLEQUIJOINcan beclassified into two types-**INNER JOIN**and**OUTER JOIN**

1. **SQLINNER JOIN**

ThistypeofEQUIJOINreturnsallrowsfromtableswherethekeyrecordofonetableis equal to the key records of another table.

2. **SQLOUTER JOIN**

This type of EQUI JOIN returns all rows from one table and only those rows from the secondarytablewherethejoinedconditionissatisfyingi.e.thecolumnsareequalinboth tables.

Inordertoperform aJOINquery,therequiredinformationweneedare:

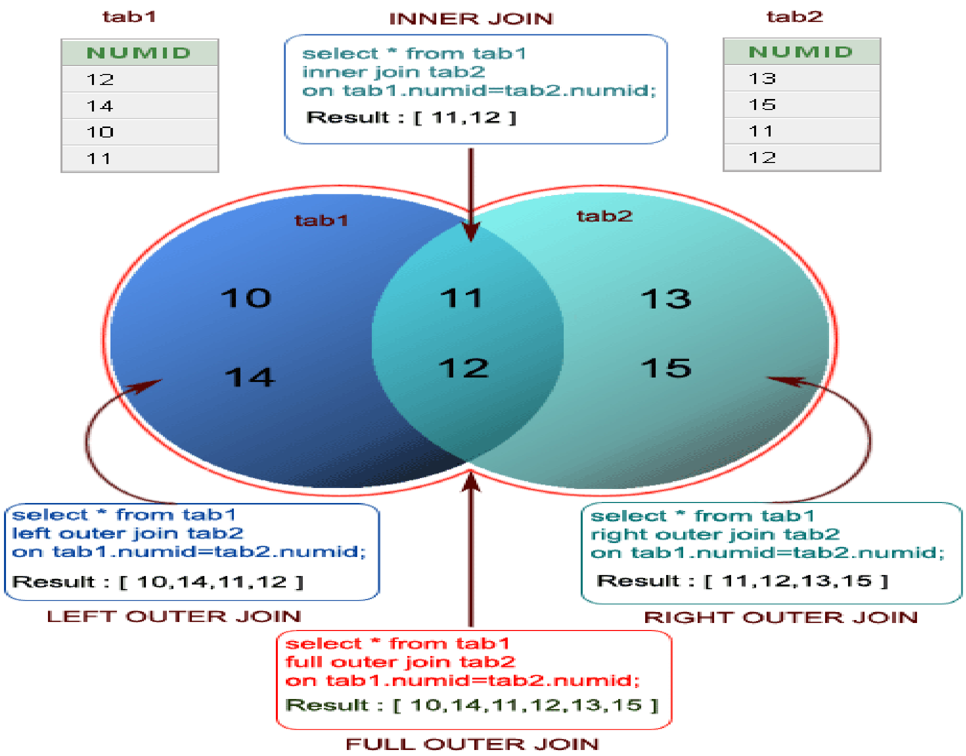
a) Thename ofthe tables

b) Nameofthecolumns of twoormoretables, based onwhich acondition will perform.

Syntax:

FROM table1
join_typetable2
[ON (join_condition)]
ONcanbereplacedwithWHERE

PictorialPresentationofSQLJoins:



Let'sConsiderthetwotablesgiven below.

Tablename- Student:

id	**Name**	**class**	**city**
3	Hina	3	Delhi
4	Megha	2	Delhi
6	Gouri	2	Delhi

Tablename—Record:

id	**Class**	**City**
9	3	Delhi
10	2	Delhi
12	2	Delhi

EQUI JOIN :

EQUIJOINcreatesaJOINforequalityor matchingcolumn(s)valuesoftherelativetables. EQUI JOIN also create JOIN by using JOIN with ON and then providing the names of the columns with their relative tables to check equality using equal sign (=).

Syntax :

SELECT column_list
FROMtable1,table2....
WHEREtable1.column_name=
table2.column_name;

Example–

SELECTstudent.name,student.id,record.class,record.city
FROM student, record
WHEREstudent.city=record.city;

Output:

name	**Id**	**class**	**City**
Hina	3	3	Delhi
Megha	4	3	Delhi
Gouri	6	3	Delhi
Hina	3	2	Delhi
Megha	4	2	Delhi
Gouri	6	2	Delhi
Hina	3	2	Delhi

name	**Id**	**class**	**City**
Megha	4	2	Delhi
Gouri	6	2	Delhi

2. NONEQUIJOIN:

NONEQUIJOINperformsaJOINusingcomparisonoperatorotherthanequal(=)signlike >,<,>=,<=withconditions.

Syntax:

SELECT*

FROMtable_name1,table_name2

WHEREtable_name1.column[>|<|>=|<=]table_name2.column;

Example–

SELECT student.name, record.id, record.city

FROM student, record

WHEREStudent.id<Record.id;

Output:

name	Id	city
Hina	9	Delhi
Megha	9	Delhi
Gouri	9	Delhi
Hina	10	Delhi
Megha	10	Delhi

name	Id	city
Gouri	10	Delhi
Hina	12	Delhi
Megha	12	Delhi
Gouri	12	Delhi

NestedQueriesinSQL:

Innestedqueries,aqueryiswritteninsideaquery.Theresultofinnerqueryisusedin execution of outer query. Nested Queries are also called as**subqueries**.

Subqueriesareusefulwhenyoumustexecutemultiplequeriestosolveasingleproblem. Each query portion of a statement is called a query block. In the following query, the subquery in parentheses is the inner query block:

```
SELECTfirst_name,last_nameFROMemployees
WHEREdepartment_id
IN    (SELECTdepartment_id
      FROM departments
      WHERElocation_id=1800);
```

- TheinnerSELECTstatementretrievestheIDsofdepartmentswithlocationID1800. These department IDs are needed by the outer query block, which retrieves names of employees in the departments whose IDs were supplied by the subquery.
- ThestructureoftheSQLstatementdoesnotforcethedatabasetoexecutetheinner query first. For example, the database could rewrite the entire query as a join of employees and departments, so that the subquery never executes by itself.

Subqueriescanbe**correlated** or**uncorrelated**.

Correlatedsubquery-Incorrelatedsubquery,innerqueryisdependentontheouterquery. Outer query needs to be executed before inner query

Non-Correlatedsubquery-Innon-correlatedqueryinnerquerydoesnotdependentonthe outer query. They both can run separately.

CorrelatedSubqueries

Acorrelatedsubquerytypicallyobtainsvaluesfromitsouterquerybeforeitexecutes.When the subquery returns, it passes its results to the outer query.

Inthefollowingexample,thesubqueryneedsvaluesfromthe**addresses.state**columninthe outer query:

=>SELECTname,street,city,stateFROM addresses
WHEREEXISTS(SELECT*FROMstatesWHEREstates.state=addresses.state); This query is executed as follows:

- Thequeryextractsandevaluateseachaddresses.statevalueintheoutersubquery records.
- Thenthequery—usingtheEXISTSpredicate—checkstheaddressesintheinner (correlated) subquery.
- BecauseitusestheEXISTSpredicate,thequerystopsprocessingwhenitfindsthe first match.

NoncorrelatedSubqueries

Anoncorrelatedsubqueryexecutesindependentlyoftheouterquery.Thesubqueryexecutes first, and then passes its results to the outer query, For example:

=>SELECTname,street,city,stateFROMaddressesWHEREstateIN(SELECTstate FROM states);

Thisqueryisexecutedasfollows:

- ExecutesthesubquerySELECT stateFROM states(in bold).
- Passesthesubqueryresults totheouter query.

Aquery'sWHEREandHAVINGclausescanspecifynoncorrelatedsubqueriesifthe subquery resolves to a single row, as shown below:

InWHERE clause
=>SELECT COUNT(*) FROM SubQ1WHERESubQ1.a=(SELECTy from SubQ2);

InHAVING clause
=>SELECTCOUNT(*)FROMSubQ1GROUP BYSubQ1.aHAVINGSubQ1.a=
(SubQ1.a& (SELECT y from SubQ2)

Aggregatefunctions:

Aggregatefunctionsoperateonvaluesacrossrowstoperformmathematicalcalculationssuch assum,average,counting,minimum/maximumvalues,standarddeviation,andestimation,as well as some non-mathematical operations.

Anaggregatefunctiontakesmultiplerows(actually,zero,one,ormorerows)asinputand produces a single output.

VariousAggregate Functions:

1. COUNT([DISTINCT]A):Thenumberof(unique)valuesintheAcolumn.
2. SUM([DISTINCT]A):Thesumofall(unique)valuesintheAcolumn.
3. AVG([DISTINCT]A):Theaverageofall(unique)valuesintheAcolumn.
4. MAX(A): Themaximum valuein theA column.
5. MIN (A):Theminimum valuein theA column.

Letus consideratable that contains thefollowingdata:

```
selectx,yfromsimpleorderbyx,y;
+- - -+-----+
|X|Y|
|-----+- - -|
|10|20|
|20|44|
|30|70|
+----+--- +
```

Theaggregate function returns oneoutputrowformultiple input rows:

```
selectsum(x)
```

```
fromsimple;
+------+
|SUM(X)|
|--------|
|    60|
+------+
```

Now let us understand each Aggregate function with a example:

Id	Name	Salary
1	A	80
2	B	40
3	C	60
4	D	70
5	E	60
6	F	Null

Count():

Count(*):Returnstotalnumberofrecords.i.e6.

Count(salary): Return numberof NonNull values overthecolumnsalary.i.e 5.

Count(DistinctSalary): Returnnumberofdistinct NonNull valuesover thecolumn salary .i.e 4.

Sum():

sum(salary): Sum allNon Nullvalues ofColumn salaryi.e., 310

sum(Distinctsalary):SumofalldistinctNon-Nullvaluesi.e., 250.

Avg():

Avg(salary)=Sum(salary)/count(salary)= 310/5

Avg(Distinctsalary) =sum(Distinctsalary)/Count(DistinctSalary)=250/4

Min():

Min(salary): Minimumvalueinthesalarycolumnexcept NULLi.e.,40.

Max(salary): Maximum valuein thesalary i.e., 80.

AggregateFunctionsandNULLValues

SomeaggregatefunctionsignoreNULLvalues.Forexample,AVGcalculatestheaverageof values 1, 5, and NULL to be 3, based on the following formula:

(1 +5) / 2 =3

IfallofthevaluespassedtotheaggregatefunctionareNULL,thentheaggregatefunction returns NULL.

Someaggregatefunctionscanbepassedmorethanonecolumn.Forexample: select count(col1, col2) from table1;

Intheseinstances,theaggregatefunctionignoresarowif anyindividualcolumnis NULL.

```
insertintot(x,y)values
(1,2),-- No NULLs.
(3,null),--OnebutnotallcolumnsareNULL.
(null,6),--OnebutnotallcolumnsareNULL.
(null,null);-- All columns are NULL.
```

Querythe table:

```
 ---------

--------------

 ---------
```

Similarly, if SUM is called with an expression that references two or more columns, and if oneormoreofthosecolumnsisNULL,thentheexpressionevaluatestoNULL,andtherow is ignored:

```
selectsum(x+y)fromt;
+--------+
```

- - - - - - - -

SQLalsoprovidesaspecialcomparisonoperatorISNULLtotestwhetheracolumnvalueis null; for example the value of y IS NULL returns **true** when x is 3 and IS NOT NULL returns **false**.

INTRODUCTIONTO VIEWS

Aviewisatablewhoserowsarenotexplicitlystoredinthedatabasebutarecomputedas needed. Views in SQL are kind of virtual tables. A view also has rows and columns as they are in a real table in the database. We can create a view by selecting fields from one or more tables presentinthedatabase.AViewcaneitherhavealltherowsofatableorspecificrowsbased on certain condition.

SampleTables:

StudentDetails

S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

StudentMarks

ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

CREATINGVIEWS

WecancreateViewusingCREATEVIEWstatement.AViewcanbecreatedfromasingle table or multiple tables.

Syntax:

CREATEVIEWview_nameASSELECTcolumn1,column2..... FROM

table_nameWHERE condition;

view_name: Name for the View

table_name: Name of the table

condition:Conditiontoselectrows

Examples:

CreatingViewfromasingletable:

InthisexamplewewillcreateaViewnamedDetailsViewfromthetableStudentDetails. Query:

CREATEVIEWDetailsViewASSELECTNAME,ADDRESS

FROMStudentDetailsWHERES_ID<5;

Toseethe dataintheView, wecanquery theviewinthesame manneraswequeryatable.

SELECT*FROM DetailsView;

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

Creating View from multiple tables: In this example we will create a View named MarksViewfromtwotablesStudentDetailsandStudentMarks.TocreateaViewfrom multiple tables we can simply include multiple tables in the SELECT statement.

Query:

CREATEVIEWMarksViewAS

SELECTStudentDetails.NAME,StudentDetails.ADDRESS,StudentMarks.MARKS FROM

StudentDetails, StudentMarks

WHEREStudentDetails.NAME=StudentMarks.NAME;

Todisplaydata ofViewMarksView:

SELECT*FROM MarksView;

Output:

NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

DELETING VIEWS

SQLallowsustodeleteanexistingView.WecandeleteordropaViewusingtheDROP statement.

Syntax:

DROPVIEW view_name;

view_name:Nameof theView whichwewanttodelete.
Forexample,ifwe wanttodeletetheViewMarksView,we candothis as:

DROPVIEW MarksView;

UPDATING VIEWS

Therearecertainconditionsneededtobesatisfiedtoupdateaview.Ifanyoneofthese conditions is not met, then we will not be allowed to update the view.

1. TheSELECTstatementwhichisusedtocreatetheviewshouldnotincludeGROUP BY clause or ORDER BY clause.
2. TheSELECTstatement shouldnothavetheDISTINCT keyword.
3. TheView should haveall NOTNULL values.
4. Theviewshould notbecreated usingnestedqueries orcomplex queries.

5. Theviewshouldbecreatedfromasingletable.Iftheviewiscreatedusingmultiple tables then we will not be allowed to update the view.

WecanusetheCREATEORREPLACEVIEWstatementtoaddorremovefieldsfroma view.

Syntax:

CREATEORREPLACEVIEWview_nameAS
SELECT column1,coulmn2,..
FROM table_name
WHEREcondition;

Forexample,ifwewanttoupdatetheviewMarksViewandaddthefieldAGEtothisView from StudentMarks Table, we can do this as:

CREATEORREPLACEVIEWMarksViewAS
SELECTStudentDetails.NAME,StudentDetails.ADDRESS,StudentMarks.MARKS,
StudentMarks.AGEFROM StudentDetails, StudentMarks
WHEREStudentDetails.NAME=StudentMarks.NAME; If

we fetch all the data from MarksView now as:

SELECT*FROM MarksView;

Output:

NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

Insertingarowinaview:

We can insert a row in a View in a same way as we do in a table. We can use the INSERT INTO statement of SQL to insert a row in a View.**Syntax**:

INSERT INTO view_name(column1, column2 , column3,..)

VALUES(value1, value2, value3..);

view_name:NameoftheView

Example:
In the below example we will insert a new row in the View DetailsView which we have created above in the example of “creating views from a single table”.
INSERTINTODetailsView(NAME,ADDRESS)

VALUES("Suresh","Gurgaon");

If we fetch all the data from DetailsView now as,

SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

DeletingarowfromaView:
Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.**Syntax**:
DELETE FROM view_name

WHERE condition;

view_name:Nameofviewfromwherewewanttodeleterows
condition:Conditiontoselectrows
Example:
In this example we will delete the last row from the view DetailsView which we just added in the above example of inserting rows.
DELETE FROM DetailsView

WHERE NAME="Suresh";

If we fetch all the data from DetailsView now as,

SELECT * FROM DetailsView;

Output:

NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

TRIGGERS

A trigger is a stored procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database. A trigger description contains three parts:

Event: Achangetothedatabasethatactivatesthetrigger.

Condition: Aqueryortestthatisrunwhenthetriggerisactivated.

Action: Aprocedurethatisexecutedwhenthetriggerisactivatedanditscon-ditionistrue.

A **trigger action** can examine the answers to the query in the condition part of the trigger, refer to old and new values of tuples modified by the statement activating the trigger, execute new queries, and make changes to the database.

Syntax:

create trigger [trigger_name]

[before | after]

{insert | update | delete}

on [table_name]

[for each row]

[trigger_body]

Explanationofsyntax:

1. create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.
2. [before|after]:Thisspecifieswhenthetriggerwillbeexecuted.
3. {insert|update|delete}:ThisspecifiestheDMLoperation.
4. on[table_name]:Thisspecifiesthenameofthetableassociatedwiththetrigger.
5. [for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.
6. [trigger_body]:Thisprovidestheoperationtobeperformedastriggerisfired

BEFOREandAFTERofTrigger:

BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

ExamplesofTriggersinSQL

The trigger called init count initializes a counter variable before every execution of an INSERT statement that adds tuples to the Students relation. The trigger called incr count increments the counter for each inserted tuple that satisfies the condition age < 18.

CREATE TRIGGER init count BEFORE INSERT ON Students /* Event */

DECLARE

countINTEGER;

BEGIN

count:=0;

END

/*Action*/

CREATE TRIGGER incr count AFTER INSERT ON Students /* Event */

WHEN (new.age< 18) /* Condition; ‘new’ is just-inserted tuple */

FOREACHROW

BEGIN/*Action;aprocedureinOracle’sPL/SQLsyntax*/

count :=count +1;

END

(identifyingthemodifiedtable,Students,andthekindofmodifyingstatement,an INSERT), and the third field is the number of inserted Students tuples with age < 18. (The trigger in Figure 5.19 only computes the count; an additional trigger is required to insert the appropriate tuple into the statistics table.)

CREATE TRIGGER set count AFTER INSERT ON Students /* Event */

REFERENCING NEW TABLE AS InsertedTuples

FOREACHSTATEMENT

INSERT/*Action*/

INTO StatisticsTable(ModifiedTable, ModificationType, Count) SELECT

‘Students’,‘Insert’,COUNT*FROMInsertedTuplesIWHEREI.age<18

UNIT IV

DEPENDENCIESANDNORMALFORMS

Importance of a good schema design

Whatis aDatabaseSchema?

A database schema is a blueprint that represents the tables and relations of a data set. Good databaseschemadesignisessentialtomakingyourdatatractablesothatyoucanmakesense of it and build the dashboards, reports, and data models that you need.

Itisimportanttohaveagooddatabaseschemadesign.Thereasonsare:

- Withoutagooddatabasedesign,thedatabaseislikelytobeunsatisfactory.
- A good database design must be implemented in such ways that the queries are written in a simple and easier manner.
- A good database design doesn’t have data redundancies (data redundancy refers to duplication of data.).
- The accuracy must be good enough after the implementation of good database design.

Four specific issues resulting from bad schema design:

1. **ReferentialIntegrity**:apoorlydonedatabasedesignleavestheapplication vulnerable to referential integrity issues.
2. **Scalability:**apoorlydonedesignwouldstruggletoscalewhenfutureapplication functionality is added.
3. **Performance**:over-orunder-normalizationcanresultinsignificantperformance issues in the application that attempts to work with the model.
4. **Maintainability**:apoordatabasedesignwillmakelifemiserablefordevelopers attempting to code against the model or to comprehend the model to diagnose issues.

IntroductionofDatabaseNormalization

Database normalization is the process of structuring and handling the relationship between data to minimize redundancy in the relational table and avoid the unnecessary anomalies propertiesfromthedatabaselikeinsertion,updateanddelete.Ithelpstodividelargedatabase tablesintosmallertablesandmakearelationshipbetweenthem.Itcanremovetheredundant data and ease to add, manipulate or delete table fields.

A normalization defines rules for the relational table as to whether it satisfies the normal form. A**normal form** is a process that evaluates each relation against defined criteria and removesthemulti valued,joins,functionalandtrivialdependencyfrom arelation. Ifanydata is updated, deleted or inserted, it does not cause any problem for database tables and help to improve the relational table' integrity and efficiency.

Objectiveof Normalization

1. Itisusedtoremovetheduplicatedataanddatabaseanomaliesfromtherelational table.
2. Normalizationhelpstoreduceredundancyandcomplexitybyexaminingnewdata types used in the table.
3. It is helpful to divide the large database table into smaller tables and link them using relationship.
4. Itavoidsduplicatedataornorepeatinggroupsintoa table.
5. Itreducesthechancesfor anomaliestooccurin adatabase.

FunctionalDependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

ForanyrelationR,attributeYisfunctionallydependentonattributeX(usuallythePK),iffor every valid instance of X, that value of X uniquely determines the value of Y. This relationship is indicated by the representation below :

$X \rightarrow Y$

TheleftsideofFDisknownasadeterminant,therightsideoftheproductionisknownasa dependent.

For example:

Assumewehavean employeetablewith attributes:Emp_Id, Emp_Name, Emp_Address.

HereEmp_IdattributecanuniquelyidentifytheEmp_Nameattributeofemployeetable because if we know the Emp_Id, we can tell that employee name associated with it.

Functionaldependencycanbewrittenas:

Emp_Id → Emp_Name

Wecansay thatEmp_Nameis functionallydependenton Emp_Id.

A **function dependency**A → B means for all instances of a particular value of A, there is the same value of B.

For example in the below table A → B is true, but B → A is not true as there are different values of A for B = 3.

AB

- - - -

1 3

2 3

4 0

1 3

4 0

TrivialFunctionalDependency

- A→B hastrivial functional dependency ifB isasubsetofA.
- Thefollowingdependenciesarealsotriviallike:A→A,B→B

Examples

- ABC→AB
- ABC→A
- ABC→ABC

NonTrivialFunctionalDependencies

X→YisanontrivialfunctionaldependencywhenYisnotasubsetofX.

- X→YiscalledcompletelynonTrivialwhenXintersectYisNULL.

Example:

- Id→Name,
- Name→DOB

SemiNonTrivialFunctionalDependencies

X→Yiscalledseminon-trivialwhenXintersectYisnotNULL.

Examples:

- AB→BC,
- AD→DC

Armstrong'sAxiomsinFunctionalDependency

The term Armstrong axioms refer to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong, that is used to test the logical implication of **functional dependencies.**

IfFis a set offunctional dependencies then the closureof F, denoted as F^+, is theset of all functionaldependencieslogicallyimpliedbyF.Armstrong'sAxiomsareasetofrules,that when applied repeatedly, generates a closure of functional dependencies.

1. **Axiomofreflexivity–**

   If X is a set of attributes and Y is subset of X, then X holds Y.

If X ⊇ Y then X→Y

Example:

X={a,b,c,d,e} Y =
{a, b, c}

Thispropertyistrivialproperty.

2. **Axiomofaugmentation–**

   Theaugmentationisalsocalledasapartialdependency.Inaugmentation,ifXdetermines Y, then XZ determines YZ for any Z.

   IfX →Ythen XZ→YZ

Example:

ForR(ABCD),**if**A→B thenAC→BC

3. **Axiomoftransitivity–**

   In the transitive rule, if X determines Y and Y determine Z, then X must also determineZ.

   IfX→Y andY→ZthenX→Z

SecondaryRules–

Theserulescanbederivedfromtheaboveaxioms.

1. **Union–**

   Unionrulesays,ifXdeterminesYandXdeterminesZ,thenXmustalsodetermineY and Z.

   IfX →YandX→ZthenX→ YZ

2. **Decomposition–**

   Decomposition rule is also known as project rule. It is the reverse of union rule.This Rule says, if X determines Y and Z, then X determines Y and X determines Z separately.

   IfX→YZthenX→Y and X→ Z

3. **PseudoTransitivity–**

   InPseudotransitiveRule,ifXdeterminesYandYZdeterminesW,thenXZdetermines W.

   IfX→YandYZ→WthenXZ→W

MinimalCovers:

A minimal cover of a set of functional dependencies (FD) E is a minimal set of dependencies F that is equivalent to E.

Theformal definitionis:Aset ofFDFtobe minimalif itsatisfiesthefollowing conditions−

- Everydependency inFhas asingleattributeforits right-hand side.
- Wecannot replace any dependency X->Ain F with adependency Y->A, where Yis a proper subset of X, and still have a set of dependencies that is equivalent to F.
- We cannot remove any dependency from F and still have a set of dependencies thatare equivalent to F.

CanonicalcoveriscalledminimalcoverwhichiscalledtheminimumsetofFDs.AsetofFD FC is called canonical cover of F if each FD in FC is a −

- Simple FD.
- Leftreduced FD.
- Non-redundantFD.

SimpleFD−X->Yis asimpleFD if Yis a singleattribute.

LeftreducedFD−X->YisaleftreducedFDiftherearenoextraneousattributesinX.
{extraneousattributes:LetXA->Ythen,Aisaextraneousattributeif X_>Y}

Non-redundantFD−X->YisaNon-redundantFDifitcannotbederivedfromF-{X->y}. Example

ConsideranexampletofindcanonicalcoverofF.

Thegivenfunctionaldependenciesareasfollows −

A->BC

B -> C

A -> B

AB->C

- Minimal cover: The minimal cover is the set of FDs which areequivalent to the given FDs.
- Canonicalcover: Incanonicalcover,theLHS(LeftHandSide)mustbe unique.

Firstofall,wewill findtheminimalcover andthenthecanonical cover.

Firststep−ConvertRHSattributeintosingletonattribute.

A->B

A->C

B -> C

A -> B

AB->C

Secondstep−RemovetheextraLHSattribute Find

the closure of A.

A+={A, B, C}

So,AB->CcanbeconvertedintoA->C A ->

B

A->C

B->C

A->B

A->C

Thirdstep−RemovetheredundantFDs. A -

> B

A->C

Now, we will convert the above set of FDs into canonical cover.

ThecanonicalcoverfortheabovesetofFDswillbeasfollows−

A->BC

B->C

NORMAL FORMS

Givenarelationschema,weneedtodecidewhetheritisagooddesignorwhetherweneedto decompose it into smaller relations. Such a decision must be guided by an understanding of what problems, if any, arise from the current schema. To provide such guidance, several normalformshavebeenproposed.Ifarelationschemaisinoneofthesenormalforms,we know that certain kinds of problems cannot arise.

Thenormal formsbasedon FDs:

FirstNormalForm (1NF):

FirstNormalFormisdefinedinthedefinitionofrelations(tables)itself.Thisruledefinesthat all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Inthe*firstnormalform*,onlysinglevaluesarepermittedattheintersectionofeachrowand column; hence, there are no repeating groups.

Tonormalizearelationthatcontainsarepeatinggroup,removetherepeatinggroupandform two new relations.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

Were-arrangetherelation (table)asbelow, toconvertit toFirstNormal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

Eachattributemustcontain onlyasinglevalue fromitspre-defined domain.

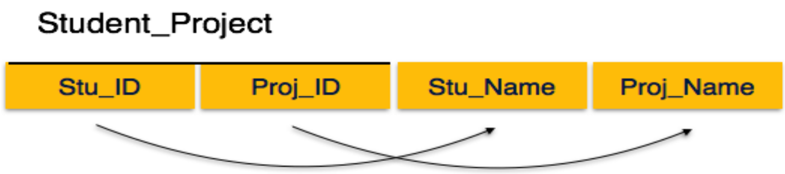
SecondNormalForm (2NF):

Beforewelearnabout thesecondnormal form, weneed to understandthefollowing −

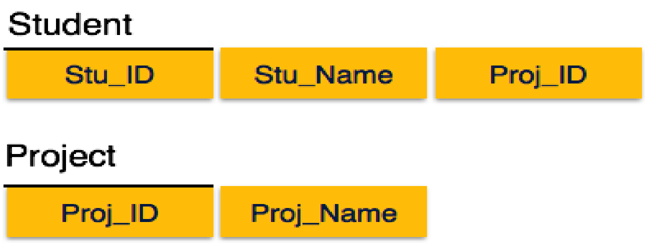
- **Prime Key attribute**− An attribute, which is a part of the candidate-key, is knownas a prime attribute.

- **Non-prime attribute**− An attribute, which is not a part of the prime-key, is said tobe a non-prime attribute.

- For the *second normal form*, the relation must first be in 1NF. The relation is automaticallyin2NFif,andonlyif,thePrimeKeycomprisesasingleattribute.
- IftherelationhasacompositePrimeKey,theneachnon-keyattributemustbefully dependent on the entire PK and not on a subset of the PK.
- Arelationisin2NFifithas**NoPartialDependency.**
- **Partial Dependency –** If the proper subset of candidate key determines non-prime attribute, it is called partial dependency.



WeseehereinStudent_ProjectrelationthattheprimekeyattributesareStu_IDandProj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name canbeidentifiedbyStu_IDandProj_NamecanbeidentifiedbyProj_IDindependently.This is called **partial dependency**, which is not allowed in Second Normal Form.



Webroketherelationintwoasdepictedintheabovepicture.Sothereexistsnopartial dependency.

ThirdNormal Form(3NF):

Tobein *thirdnormal form*,the relationmustbe insecond normalform. Also

- alltransitivedependenciesmustberemoved;anon-keyattributemaynotbefunctionally dependent on another non-key attribute.

- For any non-trivial functional dependency, X→ A, then either−
  - X is a superkey or,
  - A is prime attribute.

Transitive dependency – If A->B and B->C are two FDs then A->C is called transitive dependency.

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID→Zip→City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows −

Student_Detail

Stu_ID	Stu_Name	Zip

ZipCodes

Zip	City

Boyce-Codd Normal Form (BCNF):

Boyce-Codd Normal Form (BCNF) is an extension of Third Normal Form on strict terms.

A relation is in BCNF iff in every non-trivial functional dependency X –> Y, X is a super key.

In the above example, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So,

Stu_ID→Stu_Name,Zip and

Zip → City

Whichconfirmsthat boththerelations arein BCNF.

DECOMPOSITIONS

Arelationin BCNFis freeof redundancyand arelation schemain 3NFcomes close. Ifarelationschemaisnotinoneofthesenormalforms,theFDsthatcauseaviolationcan give us insight into the potential problems..

Adecompositionofarelationschema*R*consistsofreplacingtherelationschemaby two(ormore) relationschemasthateachcontain asubsetoftheattributes of *R*andtogether include all attributes in *R*.

When a relation in the relational model is not appropriate normal form then the decomposition of a relation is required. In a database, breaking down the table into multiple tables termed as decomposition.

Thepropertiesofarelationaldecompositionarelistedbelow:

1. **AttributePreservation:**
   Using functional dependencies the algorithms decompose the universal relation schema Rina setofrelationschemasD={R1, R2, ….. Rn}relationaldatabase schema, where 'D' is called the Decomposition of R.
   The attributes in R will appear in at least one relation schema Ri in the decomposition, i.e., no attribute is lost. This is called the *Attribute Preservation* condition of decomposition.
2. **DependencyPreservation:**
   If each functional dependency X->Y specified in F appears directly in one of therelation schemas Ri in the decomposition D or could be inferred from the dependencies that appear in some Ri. This is the *Dependency Preservation*.

If a relation R is decomposed into relation R1 and R2, then the dependencies of R either must be a part of R1 or R2 or must be derivable from the combination of functional dependencies of R1 and R2.

For example, suppose there is a relation R (A, B, C, D) with functional dependency set (A->BC). The relational R is decomposed into R1(ABC) and R2(AD) which is dependency preserving because FD A->BC is a part of relation R1(ABC).

3. LosslessJoin:

Lossless join property is a feature of decomposition supported by normalization. It is the ability to ensure that any instance of the original relation can be identified from corresponding instances in the smaller relations.

Forexample:

R : relation, F : set of functional dependencies on R,

X, Y : decomposition of R,

A decomposition {R1, R2, …, Rn} of a relation R is called a lossless decomposition for R if the natural join of R1, R2, …, Rn produces exactly the relation R.

- Therelationissaidtobelosslessdecompositionifnaturaljoinsofallthedecomposition give the original relation.

Decompositionislosslessif:

X intersection Y -> X, that is: all attributes common to both X and Y functionally determine ALL the attributes in X.

X intersection Y -> Y, that is: all attributes common to both X and Y functionally determine ALL the attributes in Y

If X intersection Y forms a superkey of either X or Y, the decomposition of R is a lossless decomposition.

4. LackofDataRedundancy

- LackofDataRedundancy is alsoknown as a**Repetitionof Information.**
- Theproperdecompositionshouldnotsufferfrom anydataredundancy.
- Thecarelessdecomposition maycause aproblemwiththedata.
- Thelack ofdata redundancyproperty maybeachievedby Normalization process.

UNIT-V

TRANSACTIONMANAGEMENT

WhatisaTransaction?

A transaction is an event which occurs on the database. Generally a transaction reads a value from the database or writes a value to the database. If you have any concept of Operating Systems, then we can say that a transaction is analogous to processes. Although a transaction can both read and write on the database, there are some fundamental differences between these two classes of operations. A read operation does not change the image of the databasein any way. But a write operation, whether performed with the intention of inserting,updating or deleting data from the database, changes the image of the database. That is, we may say that these transactions bring the database from an image which existed before the transaction occurred (called the **Before Image** or **BFIM**) to an image which exists after the transaction occurred (called the **After Image** or **AFIM**).

TheFourPropertiesof Transactions

Every transaction, for whatever purpose it is being used, has the following four properties. Taking the initial letters of these four properties we collectively call them the **ACID Properties**. Here we try to describe them and explain them.

Atomicity: This means that either all of the instructions within the transaction will be reflected in the database, or none of them will be reflected.

Say for example, we have two accounts A and B, each containing Rs 1000/-. We now start a transaction to deposit Rs 100/- from account A to Account B.

ReadA;
A =A– 100;
WriteA;
Read B;
B =B +100;
WriteB;

Fine, is not it? The transaction has 6 instructions to extract the amount from A and submit it to B.The AFIM will show Rs 900/- in A and Rs 1100/- in B.

Now, suppose there is a power failure just after instruction 3 (Write A) has been complete. What happens now? After the system recovers the AFIM will show Rs 900/- in A, but the same Rs 1000/- in B. It would be said that Rs 100/- evaporated in thin air for the power failure. Clearly such a situation is not acceptable

The solution is to keep every value calculated by the instruction of the transaction not in any stable storage (hard disc) but in a volatile storage (RAM), until the transaction completes its last instruction. When we see that there has not been any error we do something known as a **COMMIT** operation. Its job is to write every temporarily calculated value from the volatile storage on to the stable storage. In this way, even if power fails at instruction 3, the post recovery image of the database will show accounts A and B both containing Rs 1000/-, as if the failed transaction had never occurred.

Consistency: If we execute a particular transaction in isolation or together with other transaction,(i.e. presumably in a multi-programming environment), the transaction will yield the same expected result.

To give better performance, every database management system supports the execution of multiple transactions at the same time, using CPU Time Sharing. Concurrently executing transactions may have to deal with the problem of sharable resources, i.e. resources that multiple transactions are trying to read/write at the same time. For example, we may have a table or a record on which tw transaction are trying to read or write at the same time. Careful mechanisms are created in order to prevent mismanagement of these sharable resources, so that there should not be any change in the way a transaction performs. A transaction which deposits Rs 100/- to account A must deposit the same amount whether it is acting alone or in conjunction with another transaction that may be trying to deposit or withdraw some amount at the same time.

Isolation: In case multiple transactions are executing concurrently and trying to access a sharable resource at the same time, the system should create an ordering in their execution so that they should not create any anomaly in the value stored at the sharable resource.

There are several ways to achieve this and the most popular one is using some kind oflocking mechanism. Again, if you have the concept of Operating Systems, then you should remember the semaphores, how it is used by a process to make a resource busy beforestarting to use it, and how it is used to release the resource after the usage is over. Other processes intending to access that same resource must wait during this time. Locking isalmost similar.It states that atransaction must first lockthedataitem that it wishes to access, and release the lock when the accessing is no longer required. Once a transaction locks the data item, other transactions wishing to access the same data item must wait until the lock is released.

Durability: It states that once a transaction has been complete the changes it has madeshould be permanent.

As we have seen in the explanation of the Atomicity property, the transaction, if completes successfully, is committed. Once the COMMIT is done, the changes which the transactionhas made to the database are immediately written into permanent storage. So, after the transaction has been committed successfully, there is no question of any loss of information even if thepower fails. Committing atransaction guarantees that theAFIM has been reached.

There are several ways Atomicity and Durability can be implemented. One of them is called **Shadow Copy**. In this scheme a database pointer is used to point to the BFIM of thedatabase.Duringthetransaction,allthetemporarychangesarerecordedintoaShadowCopy, which is an exact copy of the original database plus the changes made by the transaction, whichistheAFIM.Now,ifthetransactionisrequiredtoCOMMIT,thenthedatabasepointer is updated to point to the AFIM copy, and the BFIM copy is discarded. On the other hand, if thetransaction is not committed,thenthe databasepointeris not updated. It keeps pointing to the BFIM, and the AFIM is discarded. This is a simple scheme, but takes a lot of memory space and time to implement.

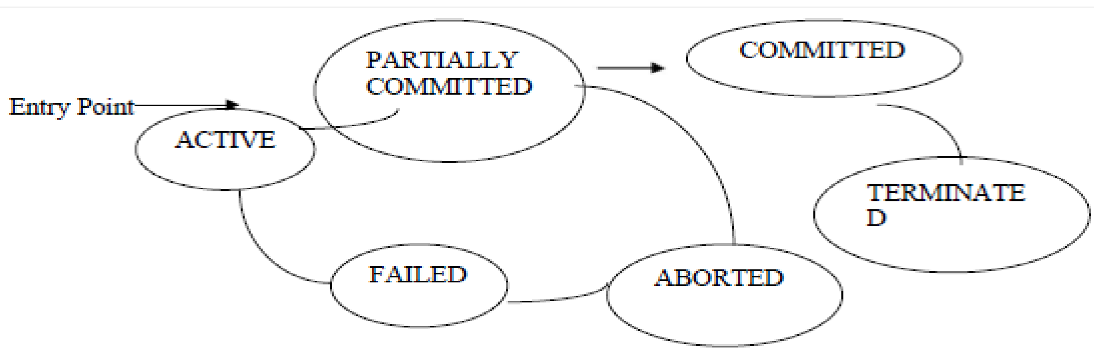
Ifyoustudycarefully,youcanunderstandthatAtomicityandDurabilityisessentiallythe same thing, just as Consistency and Isolation is essentially the same thing.

TransactionStates

Therearethefollowingsixstates inwhichatransactionmay exist:

- **Active:**Theinitialstatewhenthetransactionhasjuststartedexecution.
- **Partially Committed:** At any given point of time if the transaction is executing properly, then it is going towards it COMMIT POINT. The values generated during the execution are all stored in volatile storage.
- **Failed:** If the transaction fails for some reason. The temporary values are no longer required,andthetransactionissetto**ROLLBACK**.Itmeansthatanychangemadeto the database by this transaction up to the point of the failure must be undone. If the failed transaction has withdrawn Rs. 100/- from account A, then the ROLLBACK operation should add Rs 100/- to account A.
- **Aborted:** When the ROLLBACK operation is over, the database reaches the BFIM. The transaction is now said to have been aborted.
- **Committed:** If no failure occurs then the transaction reaches the COMMIT POINT. All the temporary values are written to the stable storage and the transaction is said to have been committed.
- **Terminated:**Eithercommittedor aborted

Thewholeprocess canbedescribed usingthe following diagram:



ConcurrentExecution

A schedule is a collection of many transactions which is implemented as a unit. Depending upon how these transactions are arranged in within a schedule, a schedule can be of twotypes:

- **Serial:**Thetransactionsareexecutedoneafteranother, inanon-preemptive manner.
- **Concurrent:**Thetransactionsareexecutedinapreemptive,timeshared method.

In Serial schedule, there is no question of sharing a single data item among manytransactions, because not more than a single transaction is executing at any point of time. However, a serial schedule is inefficient in the sense that the transactions suffer for having a longer waiting time and response time, as well as low amount of resource utilization.

In concurrent schedule, CPU time is shared among two or more transactions in order to run them concurrently. However, this creates the possibility that more than one transaction may need to access a single data item for read/write purpose and the database could contain inconsistent value if such accesses are not handled properly. Let us explain with the help ofan example.

Let us consider there are two transactions T1 and T2, whose instruction sets are given as following. T1 is the same as we have seen earlier, while T2 is a new transaction.

<u>T1</u>

ReadA;

A =A– 100;

WriteA;

Read B;

B =B +100;

WriteB;

<u>T2</u>

ReadA;

Temp=A*0.1; Read

C;

C=C+Temp; Write

C;

T2isanew transactionwhichdeposits toaccountC 10%oftheamountinaccount A.

If we prepare a serial schedule, then either T1 will completely finish before T2 can begin, or T2 will completely finish before T1 can begin. However, if we want to create a concurrent schedule, then some Context Switching need to be made, so that some portion of T1 will be executed, then some portion of T2 will be executed and so on. For example say we have prepared the following concurrent schedule.

T1	T2
Read A;	
A = A – 100;	
Write A;	
	Read A;
	Temp = A * 0.1;
	Read C;
	C = C + Temp;
	Write C;
Read B;	
B = B + 100;	
Write B;	

No problem here. We have made some Context Switching in this Schedule, the first one after executing the third instruction of T1, and after executing the last statement of T2. T1 first deducts Rs 100/- from A and writes the new value of Rs 900/- into A. T2 reads the value ofA, calculates the value of Temp to be Rs 90/- and adds the value to C. The remaining part of T1 is executed and Rs 100/- is added to B.

Serializability

When several concurrent transactions aretrying to access the same data item, the instructions within these concurrent transactions must be ordered in some way so as there are no problem in accessing and releasing the shared data item. There are two aspects of serializability which are described here:

ConflictSerializability

Two instructions oftwo different transactions may want to access the samedata item in order to perform a read/write operation. Conflict Serializability deals with detecting whether the instructionsareconflictinginanyway,andspecifyingtheorderinwhichthesetwo instructions will be executed in case there is any conflict. A **conflict** arises if at least one (or both)oftheinstructionsisawriteoperation.ThefollowingrulesareimportantinConflict Serializability:

1. Iftwoinstructionsofthetwoconcurrenttransactionsarebothforreadoperation,then they are not in conflict, and can be allowed to take place in any order.

2. If one of the instructions wants to perform a read operation and the other instruction wants to performawriteoperation, thentheyarein conflict, hencetheirorderingis important. Ifthe read instruction is performed first, then it reads the old value of the data item and after the reading is over, the new value of the data item is written. It the write instruction is performed first, then updates the data item with the new value and the read instruction reads the newly updated value.

3. If both the transactions are for write operation, then they are in conflict but can be allowed to take place in any order, because the transaction do not read the value updated by each other. However, the value that persists in the data item after the schedule is over is the one written by the instruction that performed the last write.

ViewSerializability:

This is another type of serializability that can be derived by creating another schedule out of an existing schedule, involving the same set of transactions. These two schedules would be called View Serializable if the following rules are followed while creating the second schedule out of the first. Let us consider that the transactions T1 and T2 are being serializedto create two different schedules S1 and S2 which we want to be **View Equivalent** and both T1 and T2 wants to access the same data item.

1. If in S1, T1 reads the initial value of the data item, then in S2 also, T1 should read the initial value of that same data item.

2. IfinS1,T1writesavalueinthedataitemwhichisreadbyT2,theninS2also,T1should write the value in the data item before T2 reads it.

3. If in S1, T1 performs the final write operation on that data item, then in S2 also, T1 should perform the final write operation on that data item. Let us consider a schedule *S* in which there are two consecutive instructions, *I* and *J* , of transactions *Ti* and *Tj* , respectively (*i* _=*j*). If *I* and *J* refer to different data items, then we can swap *I* and *J* without affecting the results of any instruction
in the schedule. However, if *I* and *J* refer to the same data item *Q*, then the order of the two steps may matter. Since we are dealing with only read and write instructions, there are four cases that we need to consider:

 I=read(*Q*),*J*=read(*Q*).Theorderof*I*and*J*doesnotmatter,sincethesamevalue of *Q* is read by *Ti* and *Tj* , regardless of the order.

 I = read(*Q*), *J* = write(*Q*). If *I* comes before *J* , then *Ti* does not read the value of *Q* that is written by *Tj* in instruction *J* . If *J* comes before *I*, then *Ti* reads the value of *Q* that is written by *Tj*. Thus, the order of *I* and *J* matters.

 I=write(*Q*),*J*=read(*Q*).Theorderof*I*and*J*mattersforreasonssimilartothoseofthe previous case.

4. *I* = write(*Q*), *J* = write(*Q*). Since both instructions are write operations, the order of these instructionsdoesnotaffecteither*Ti*or*Tj*.However,thevalueobtainedbythenextread(*Q*) instruction of *S* is affected, since the result of only the latter of the two write instructions is preserved in thedatabase. Ifthereis no other write(*Q*) instruction after*I*and *J*in *S*, then the order of *I* and *J* directly affects the final value of *Q* in the database state that results from schedule *S*.

T_1	T_2
read(A)	
write(A)	
	read(A)
	write(A)
read(B)	
write(B)	
	read(B)
	write(B)

Fig: Schedule 3—showing only the read and write instructions.

We say that *I* and *J* **conflict** if they are operations by different transactions on the same data item, and at least one of these instructions is a write operation. To illustrate the concept of conflicting instructions, we consider schedule 3in Figure above. The write(*A*) instruction of *T*1 conflicts with the read(*A*) instruction of *T*2. However, the write(*A*) instruction of *T*2 does not conflict with the read(*B*) instruction of *T*1, because the two instructions access different data items.

Transaction Characteristics

Every transaction has three characteristics: *access mode*, *diagnostics size*, and *isolation level*. The **diagnostics size** determines the number of error conditions that can be recorded.

If the **access mode** is READ ONLY, the transaction is not allowed to modify the database. Thus, INSERT, DELETE, UPDATE, and CREATE commands cannot be executed. If we have to execute one of these commands, the access mode should be set to READ WRITE. F transactions with READ ONLY access mode, only shared locks need to be obtained, thereby increasing concurrency.

The **isolation level** controls the extent to which a given transaction is exposed to the actions of other transactions executing concurrently. By choosing one of four possible isolation level settings, a user can obtain greater concurrency at the cost of increasing the transaction's exposure to other transactions' uncommitted changes.

Isolationlevelchoicesare READUNCOMMITTED,READCOMMITTED,REPEATABLE READ, and SERIALIZABLE. The effect of these levels is summarized in Figure given below.Inthiscontext,*dirtyread*and*unrepeatableread*aredefinedasusual.**Phantom**is defined to be the possibility that a transaction retrieves a collection of objects (in SQL terms, a collection of tuples) twice and sees different results, even though it does not modify any of these tuples itself.

In terms of a lock-based implementation, a SERIALIZABLE transaction obtains locks before readingorwritingobjects,includinglocksonsetsofobjectsthatitrequirestobeunchanged ,andholdsthemuntiltheend,accordingtoStrict2PL. **REPEATABLEREAD**ensuresthat*T* reads only the changes made by committed transactions, and that no value read or written by *T* is changed by any other transaction until *T* is complete. However, *T* could experience the phantom phenomenon; for example, while *T* examines all Sailors records with *rating=1*, another transaction might add a new such Sailors record, which is missed by *T*.

A REPEATABLE READ transaction uses the same locking protocol as a SERIALIZABLE transaction, except that it does not do index locking, that is, it locks only individual objects, not sets of objects.

READ COMMITTED ensures that *T* reads only the changes made by committed transactions, and that no value written by *T* is changed by any other transaction until *T* is complete.However,avaluereadby*T*maywellbemodifiedby anothertransactionwhile *T*is still in progress, and *T* is, of course, exposed to the phantom problem.

A READ COMMITTED transaction obtains exclusive locks before writing objects and holds these locks until the end. It also obtains shared locks before reading objects, but these locks are released immediately; their only effect is to guarantee that the transaction that last modified the object is complete. (This guarantee relies on the fact that *every* SQL transaction obtains exclusive locks before writing objects and holds exclusive locks until the end.)

AREADUNCOMMITTEDtransactiondoesnotobtainsharedlocksbeforereadingobjects. This mode represents the greatest exposure to uncommitted changes of other transactions; so much so that SQL prohibits such a transaction from making any changes itself - a READ UNCOMMITTED transaction is required to have an access mode of READ ONLY. Since such a transaction obtains no locks for reading objects, and it is not allowed to write objects (and therefore never requests exclusive locks), it never makes any lock requests

The SERIALIZABLE isolation level is generally the safest and is recommended for most transactions. Some transactions, however, can run with a lower isolation level, and thesmaller number of locks requested can contribute to improved system performance.

For example, a statistical query that finds the average sailor age can be run at the READ COMMITTED level, or even the READ UNCOMMITTED level, because a few incorrect or missing values will not significantly affect the result if the number of sailors is large. The isolation level and access mode can be set using the SET TRANSACTION command. For example, the following command declares the current transaction to be SERIALIZABLE and READ ONLY:

SETTRANSACTIONISOLATIONLEVELSERIALIZABLEREADONLY

WhenatransactionisStarted,thedefaultisSERIALIZABLEandREAD WRITE.

PRECEDENCEGRAPH.

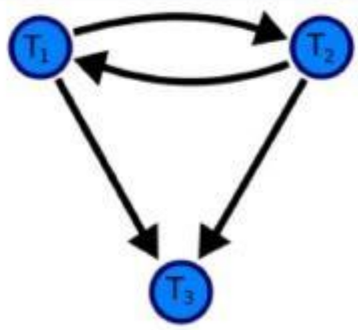
Aprecedencegraph,alsonamedconflictgraphandserializabilitygraph,isusedinthe context of concurrency control in databases.

Theprecedencegraph for ascheduleS contains:

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

Precedence graph example

$$D = \begin{bmatrix} T1 & T2 & T3 \\ & R(B) & \\ R(C) & W(A) & \\ W(C) & & \\ R(D) & & \\ & & W(B) \\ W(D) & & W(A) \\ & & \end{bmatrix}$$

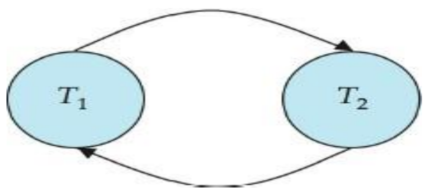


A precedence graph of the schedule D, with 3 transactions. As there is a cycle (of length 2; with two edges) through the committed transactions T1 and T2, this schedule (history) is not Conflict serializable.

Thedrawing sequencefortheprecedencegraph:-

- For each transaction T_i participating in schedule S, create a node labelled T_i in the precedence graph. So the precedence graph contains T_1, T_2, T_3
- Foreach case in S where $T_i$executes a write_item(X) then $T_j$executes a read_item(X), create an edge (T_i --> T_j) in the precedence graph. This occurs nowhere in the above example, as there is no read after write.
- ForeachcaseinSwhereT_i executesaread_item(X)thenT_j executesawrite_item(X), create an edge (T_i--> T_j) in the precedence graph. This results in directed edge from T_1 to T_2.
- For each case in S where $T_i$executes a write_item(X) then $T_j$executes a write_item(X), create an edge (T_i --> T_j) in the precedence graph. This results in directed edges from T_2 to T_1, T_1 to T_3, and T_2 to T_3.

- The schedule S is conflict serializable if the precedence graph has no cycles. As T_1 and T_2 constitute a cycle, then we cannot declare S as serializable or not and serializability has to be checked using other methods.



TESTINGFORCONFLICTSERIALIZABILITY

- 1Aschedule isconflictserializable if andonly ifitsprecedencegraphis acyclic.
- 2 To test for conflict serializability, we need to construct the precedence graph and to invoke a cycle-detection algorithm.Cycle-detection algorithms exist which take order n2 time, where n is the number of vertices in the graph. (Better algorithms take ordern + e where e is the number of edges.)
- 3 If precedence graph is acyclic, the serializability order can be obtained by a topologicalsortingofthegraph.Thatis,alinearorderconsistentwiththepartialorder of the graph.

  For example, a serializability order for the schedule (a) would be one of either (b) or (c)
- A serializability order of the transactions can be obtained by finding a linear order consistent with the partial order of the precedence graph.

RECOVERABLE SCHEDULES

- Recoverable schedule — if a transaction Tj reads a data item previously written by a transaction Ti , then the commit operation of Ti must appear before the commit operation of Tj.
- The following schedule is not recoverable if T9 commits immediately after the read(A) operation.

T_8	T_9
read (A)	
write (A)	
	read (A)
	commit
read (B)	

- If T8 should abort, T9 would have read (and possibly shown to the user) an inconsistent database state. Hence, database must ensure that schedules are recoverable.

CASCADINGROLLBACKS

- Cascading rollback – a single transaction failure leads to a series of transaction rollbacks. Consider the following schedule where none of the transactions has yet committed (so the schedule is recoverable) If T10 fails, T11 and T12 must also be rolled back.
- Canlead totheundoing ofasignificantamount of work

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

CASCADELESS SCHEDULES

- Cascade less schedules — for each pair of transactions Ti and Tj such that Tj reads a dataitem previously writtenby Ti, thecommit operation ofTi appearsbeforetheread operation of Tj.
- Everycascadeless scheduleisalso recoverable
- Itisdesirable torestrict theschedulesto thosethatarecascadeless.
- Exampleof aschedule that isNOT cascadeless

T_{10}	T_{11}	T_{12}
read (A)		
read (B)		
write (A)		
	read (A)	
	write (A)	
		read (A)
abort		

CONCURRENCYSCHEDULE

Adatabasemustprovide amechanismthatwillensurethat allpossibleschedulesare both:

- Conflictserializable.
- Recoverableandpreferablycascadeless
- Apolicyinwhichonlyonetransactioncanexecuteatatimegeneratesserial schedules, but provides a poor degree of concurrency.
- Concurrency-control schemes tradeoff between theamount of concurrency they allow and the amount of overhead that they incur
- Testingaschedule forserializabilityafterithasexecutedisalittletoo late!
- Testsforserializabilityhelpusunderstandwhyaconcurrencycontrolprotocolis correct
- Goal–todevelopconcurrency controlprotocolsthatwillassureserializability.

WEEKLEVELSOFCONSISTENCY

- Some applications are willing to live with weak levels of consistency, allowingschedules that are not serializable
- E.g.,aread-onlytransactionthatwantstogetanapproximatetotalbalanceofall accounts
- E.g.,databasestatistics computedforqueryoptimizationcan be approximate (why?)
- Suchtransactionsneednotbeserializable withrespecttoothertransactions
- Tradeoffaccuracyforperformance

LEVELSOFCONSISTENCYINSQL

- Serializable— default
- Repeatable read — only committed records to be read, repeated reads of same record must return same value. However, a transaction may not be serializable – it may find some records inserted by a transaction but not find others.
- Read committed — only committed records can be read, but successive reads of record may return different (but committed) values.
- Readuncommitted —evenuncommittedrecordsmayberead.
- Lower degrees of consistency useful for gathering approximate information about the database
- Warning:somedatabase systemsdonotensureserializableschedulesby default
- E.g.,OracleandPostgreSQLbydefaultsupportalevelofconsistencycalledsnapshot isolation (not part of the SQL standard)

TRANSACTIONDEFINITIONIN SQL

- Data manipulation language must include a construct for specifying the set of actions that comprise a transaction.
- InSQL,atransactionbeginsimplicitly.
- Atransaction inSQLends by:
- Commitworkcommitscurrenttransactionandbeginsanewone.
- Rollbackworkcausescurrenttransactiontoabort.
- Inalmostalldatabasesystems,bydefault,everySQLstatementalsocommits implicitly if it executes successfully
- Implicitcommitcanbeturned offbyadatabasedirective
- E.g.inJDBC, connection. setAuto Commit(false);

RECOVERY SYSTEM

FailureClassification:

- Transactionfailure:
- Logicalerrors:transactioncannotcompleteduetosomeinternalerror condition
- Systemerrors:thedatabasesystemmustterminateanactivetransactionduetoan error condition (e.g., deadlock)

- System crash: a power failure or other hardware or software failure causes the system to crash.
- Fail-stopassumption:non-volatilestoragecontentsareassumedtonotbecorruptedas result of a system crash
- Databasesystemshavenumerousintegritycheckstopreventcorruptionof disk data
- Diskfailure:a headcrashor similardisk failuredestroys allorpart ofdisk storage
- Destructionisassumedtobedetectable:diskdrivesusechecksumstodetectfailures

RECOVERY ALGORITHMS

- ConsidertransactionTithattransfers$50fromaccountAtoaccountB
- Two updates: subtract 50 from A and add 50 to B Transaction Ti requires updates toA and B to be output to the database.
- A failure may occur after one of these modifications have been made but before both of them are made.
- Modifying the database without ensuring that the transaction will commit may leave the database in an inconsistent state
- Notmodifyingthedatabasemayresultinlostupdatesiffailureoccursjustafter transaction commits
- Recoveryalgorithmshavetwoparts
-

1. Actionstakenduringnormaltransactionprocessingtoensureenoughinformation
exists to recover from failures

2. Actionstakenafterafailuretorecoverthedatabasecontentstoastatethatensures
atomicity, consistency and durability

STORAGE STRUCTURE

- Volatilestorage:
- Doesnotsurvivesystemcrashes
- Examples:mainmemory,cache memory
- Nonvolatilestorage:
- Survivessystemcrashes
- Examples:disk,tape,flashmemory,
- Non-Volatile(batterybackedup)RAM

- Butmay stillfail, losingdata
- Stablestorage:
- amythical formof storagethat survivesall failures
- Approximatedbymaintaining multiplecopiesondistinctnonvolatilemedia

Stable-StorageImplementation

- Maintainmultiplecopies ofeachblock onseparatedisks
- copiescanbeatremotesitestoprotectagainstdisasterssuchasfireor flooding.
- Failureduringdatatransfercan stillresultininconsistent copies.
- Blocktransfercanresultin
- Successfulcompletion
- Partialfailure:destinationblockhasincorrectinformation
- Totalfailure:destination blockwas neverupdated
- Protectingstoragemedia fromfailureduringdatatransfer(one solution):
- Executeoutputoperationsasfollows(assumingtwocopiesofeachblock):

1. Writetheinformation ontothefirstphysicalblock.
2. Whenthefirstwritesuccessfullycompletes,writethesameinformationontothe second physical block.
3. Theoutputis completedonly afterthesecond writesuccessfully completes.

- Copiesofablockmaydifferduetofailureduringoutputoperation.Torecoverfrom failure:First find inconsistent blocks:

1. Expensivesolution:Comparethetwocopiesof everydiskblock.
2. Better solution:

- Record in-progress disk writes on non-volatile storage (Non-volatile RAM or special area of disk).
- Usethisinformationduringrecoverytofindblocksthatmaybeinconsistent,and only compare copies of these.
- UsedinhardwareRAID systems

Ifeithercopy of an inconsistent blockis detected to have an error(bad checksum), overwrite itbytheothercopy.Ifbothhavenoerror,butaredifferent,overwritethesecondblock bythe first block.

DATAACCESS

- Physicalblocks arethoseblocksresiding onthedisk.
- Systembufferblocksaretheblocksresidingtemporarilyinmain memory.
- Block movements between disk and main memory are initiated through the following two operations:
- input(B)transfersthephysical blockBtomain memory.
- output(B)transfersthebufferblockBtothedisk,andreplacestheappropriatephysical block there.
- Weassume,forsimplicity,thateachdataitemfitsin,andisstoredinside,asingle block.
- Each transaction Ti has its private work-area in which local copies of all data itemsaccessed and updated by it are kept.
- Ti'slocal copy ofadata item X is denoted by xi.
- BXdenotes blockcontaining X
- Transferring data items between system buffer blocks and its private work-area done by:
- read(X)assigns thevalue of data itemX to thelocal variablexi.
- write(X) assignsthevalueof localvariable xito dataitem {X}in thebuffer block.
- Transactions
- Must perform read(X) before accessing X for the first time (subsequent reads can be from local copy)
- Thewrite(X) canbe executedat anytimebeforethetransaction commits
- Notethatoutput(BX)neednotimmediatelyfollowwrite(X).Systemcan performthe output operation when it seems fit.

Lock-Based Protocols

Alockisamechanismtocontrolconcurrentaccesstoadataitem Data items
can be locked in two modes :

1. exclusive(X)mode.Dataitemcanbebothreadaswellaswritten.X-lockisrequested using lock-X instruction.

2. shared (S) mode. Data item can only be read. S-lock is requested using lock-S instruction.
Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

Lock-compatibilitymatrix

	S	X
S	true	false
X	false	false

1) Atransactionmaybegrantedalockonanitemiftherequestedlockiscompatiblewith locks already held on the item by other transactions

2) Any number of transactions can hold shared locks on an item, but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.

3) If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

Exampleofatransactionperforminglocking:

T_2:

lock-S*(A)*;

read *(A)*;

unlock*(A)*;

lock-S*(B)*;

read *(B)*;

unlock*(B)*;

display*(A+B)*

Lockingasaboveisnotsufficienttoguaranteeserializability —if*A*and*B*getupdated in-between the read of *A* and *B*, the displayed sum would be wrong.

A**lockingprotocol**isasetofrulesfollowedbyalltransactionswhilerequestingandreleasing locks. Locking protocols restrict the set of possible schedules.

Considerthepartialschedule

T_3	T_4
lock-x (*B*)	
read (*B*)	
$B := B - 50$	
write (*B*)	
	lock-s (*A*)
	read (*A*)
	lock-s (*B*)
lock-x (*A*)	

Neither T_3 nor T_4 canmakeprogress—executing **lock-S***(B)*causesT_4 towaitforT_3 to release its lock on *B*, while executing **lock-X***(A)* causes T_3 to wait for T_4 to release its lock on *A*. Such a situation is called a **deadlock**.

l.To handleadeadlockoneofT_3orT_4mustberolledback andits locks released.

2. Thepotentialfordeadlockexistsinmostlockingprotocols.Deadlocks areanecessaryevil.

3. **Starvation**isalsopossibleifconcurrencycontrolmanagerisbadlydesigned.For example:

- Atransactionmaybewaitingforan X-lockonanitem,whileasequenceofother transactions request and are granted an S-lock on the same item.
- Thesametransactionisrepeatedlyrolledbackduetodeadlocks.

4.Concurrency control manager can be designed to prevent starvation.

THETWO-PHASELOCKINGPROTOCOL

1.Thisisaprotocolwhichensuresconflict-serializable schedules.

2.Phase1:Growing Phase

- transactionmayobtainlocks
- transactionmaynotreleaselocks

3.Phase2: Shrinking Phase

- transactionmayrelease locks
- transactionmay notobtain locks

4.The protocol assures serializability. It can be proved that the transactions can be serialized in the order of their **lock points** (i.e. the point where a transaction acquired its final lock).

5.Two-phaselocking *doesnot*ensurefreedom from deadlocks

6.Cascading roll-back is possible under two-phase locking. To avoid this, follow a modified protocol called **strict two-phase locking**. Here a transaction must hold all its exclusive locks till it commits/aborts.

7. **Rigoroustwo-phaselocking**isevenstricter:here*all*locksareheldtillcommit/abort. In this protocol transactions can be serialized in the order in which they commit.

8.Therecanbeconflictserializableschedulesthatcannotbeobtainediftwo-phase locking is used.

9.However,inthe absenceofextrainformation(e.g., orderingof access todata), twophase lockingisneeded forconflictserializabilityinthe followingsense: Given a transaction T_i that does not follow two-phase locking, we can find a transaction T_j that uses two-phase locking, and a schedule for T_i and T_j that is not conflict serializable.

TIMESTAMP-BASEDPROTOCOLS

1. Eachtransactionisissuedatimestampwhenitentersthesystem. Ifanoldtransaction $T_i$has time-stampTS(T_i),anewtransaction$T_j$isassignedtime-stampTS(T_j)suchthatTS(T_i) <TS(T_j).

2. Theprotocolmanagesconcurrentexecutionsuchthatthetime-stampsdeterminethe serializability order.

3. Inordertoassuresuchbehavior,theprotocolmaintainsforeachdata Qtwotimestamp values:

- W-timestamp(Q) is the largest time-stamp of any transaction that executed write(Q)successfully.
- R-timestamp(Q)isthelargesttime-stampofanytransactionthatexecutedread(Q) successfully.

4. Thetimestamporderingprotocolensuresthat anyconflicting **read** and **write**operations are executed in timestamp order.

5. SupposeatransactionT$_i$issues a**read**(Q)

- IfTS(T_i) □ **W**-timestamp(Q),then$T_i$needstoreadavalueof$Q$thatwasalready overwritten. n Hence, the **read** operation is rejected, and T_i is rolled back.
- IfTS(T_i) □ **W**-timestamp(Q),thenthe**read**operationisexecuted,andRtimestamp(Q)issetto**max**(R-timestamp(Q),TS(T_i)).

6. Supposethattransaction $T_i$issues**write**(Q).

- If TS(T_i) < R-timestamp(Q), then the value of Q that T_i is producing was needed previously, and the system assumed that that value would never be produced. nHence, the **write** operation is rejected, and T_i is rolled back.
- If TS(T_i) < W-timestamp(Q), then T_i is attempting to write an obsolete value of Q. n Hence, this **write** operation is rejected, and T_i is rolled back.
- Otherwise,the**write**operationisexecuted,andW-timestamp(Q)is setto TS(T_i).

1. Wenowpresentamodificationtothetimestamp-orderingprotocolthatallowsgreater potentialconcurrencythandoestheprotocoli.e.,TimestamporderingProtocol.Letus consider schedule 4 of Figure below, and apply the timestamp-ordering protocol. Since *T*27 starts before *T*28, we shall assume that TS(*T*27) <TS(*T*28). The read(*Q*) operation of *T*2 succeeds, as does the write(*Q*) operation of *T*28. When *T*27 attempts its write(*Q*) operation, we find that TS(*T*27) <W-timestamp(*Q*), since Wtimestamp(*Q*) = TS(*T*28). Thus, the write(*Q*) by *T*27 is rejected and transaction *T*27 must be rolled back.

2. Although the rollback of *T*27 is required by the timestamp-ordering protocol, it is unnecessary. Since *T*28 has already written *Q*, the value that *T*27 is attempting to write is one that will never need to be read. Any transaction *Ti* with TS(*Ti*) <TS(*T*28) that attempts a read(*Q*)will be rolled back, since TS(*Ti*)<W-timestamp(*Q*).

3. Any transaction *Tj* with TS(*Tj*) >TS(*T*28) must read the value of *Q* written by *T*28, rather thanthevaluethat*T*27is attemptingtowrite. Thisobservationleadsto amodifiedversion,of the timestamp-ordering protocol in which obsolete write operations can be ignored under certaincircumstances.Theprotocolrulesforread operationsremainunchanged. Theprotocol rules for write operations, however, are slightly different from the time stamp order in protocol.

T_{27}	T_{28}
read(*Q*)	
	write(*Q*)
write(*Q*)	

Themodificationtothetimestamp-orderingprotocol,called**Thomas'writerule**,isthis:

Suppose that transaction *Ti* issues write(*Q*).

1. If TS(*Ti*) <R-timestamp(*Q*), then the value of *Q* that *Ti* is producing was previously needed, and it had been assumed that the value would never be produced. Hence, the system rejects the write operation and rolls *Ti* back.

2. IfTS(*Ti*)<W-timestamp(*Q*),then*Ti*isattemptingtowriteanobsoletevalueof*Q*.Hence, this write operation can be ignored.

3. Otherwise,thesystemexecutesthe writeoperationandsetsW-timestamp(*Q*)toTS(*Ti*).

VALIDATION-BASEDPROTOCOLS

1) PhasesinValidation-BasedProtocolsofthevariousdataitemsandstorestheminvariables local to Ti. It performs all write operations on temporary local variables, without updates of the actual database.

2) Validation phase. The validation test is applied to transaction Ti. This determines whether Ti is allowed to proceed to the write phase without causing a violation of serializability. If a transaction fails the validation test, the system aborts the transaction.

3) Writephase.IfthevalidationtestsucceedsfortransactionTi,thetemporarylocal variables that hold the results of any write operations performed by Ti are copied to the database.Read-only transactions omit this phase.

MODESINVALIDATION-BASEDPROTOCOLS

1. Start(Ti)

2. Validation(Ti)

3. Finish

MULTIPLEGRANULARITY.

multiple granularity locking (MGL) is a locking method used in database management systems(DBMS)andrelationaldatabases. InMGL,locksaresetonobjectsthatcontainother objects.MGLexploitsthehierarchicalnatureofthecontainsrelationship.Forexample,a databasemayhavefiles,whichcontainpages,whichfurthercontainrecords.Thiscanbe thoughtofasatreeofobjects,whereeachnodecontainsitschildren.Alockonsuchasa shared or exclusive lock locks the targeted node as well

as all of its descendants. Multiple granularity locking is usually used with non-strict two-phase locking to guarantee serializability. The **multiple-granularity locking protocol** uses theselockmodestoensureserializability.Itrequiresthatatransaction *Ti*thatattemptstolock a node *Q* must follow these rules:

- Transaction*Ti*mustobservethe lock-compatibilityfunctionofFigureabove.
- Transaction*Ti* mustlock theroot ofthe treefirst,and canlock itin anymode.
- Transaction*Ti* canlocka node*Q*inSor ISmode only if*Ti*currently hastheparent of *Q*locked in eitherIXorIS mode.
- Transaction *Ti* can lock a node *Q* in X, SIX, or IX mode only if *Ti* currently has theparent of *Q* locked in either IX or SIX mode.
- Transaction *Ti* can lock a node only if *Ti* has not previously unlocked any node (that is, *Ti* is two phase).
- Transaction*Ti*canunlock anode*Q*onlyif*Ti* currentlyhasnoneofthechildren of *Q*locked.