UNIT I

DATA MODELS AND QUERYING

PURPOSE OF DATABASE SYSTEMS-VIEWS OF DATA-DATA MODELS-DATABASE SYSTEM ARCHITECTURE-INTRODUCTION TO RELATIONAL DATABASES-RELATIONAL MODEL-KEYS-RELATIONAL ALGEBRA-SQL FUNDAMENTALS-ADVANCED SQL FEATURES-EMBEDDED SQL-DYNAMIC SQL

INTRODUCTION

A **database-management system (DBMS)** is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the **database**, contains information relevant to an enterprise.

The primary goal of a DBMS is to provide a way to store and retrieve database information that is both *convenient* and *efficient*.

Database-System Applications.

- **Banking-**For customer information, account activities, payments, deposits, loans etc**.**
- **Airlines-**For reservations and schedule information.
- **Universities-** For student information, course registrations, colleges and grades.
- **Telecommunications-**It helps to keep call records, monthly bills maintaining balances etc.
- **Finance:** For storing information about stock, sales and purchases of financial instruments like stocks and bonds.
- **Accounting:** Database systems are used in maintaining information employees, salaries and payroll taxes
- **Manufacturing:** For management of supply chain and tracking production of items in factories database systems are maintained.
- **Reservations Systems:** In airline/railway reservation systems, the database is used to maintain the reservation and schedule information.

1.1 PURPOSE OF DATABASE SYSTEMS

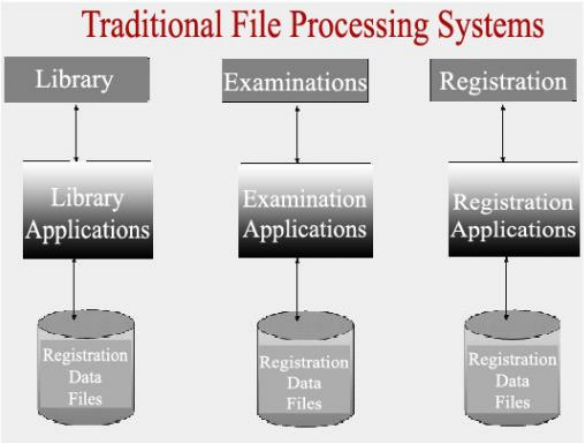
To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:

- Add new records

- Update the content in the records
- Delete the records

File Processing System

- 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 database management 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
    - ✓ Difficulty in accessing data
    - ✓ Data isolation
    - ✓ Integrity problems
    - ✓ Atomicity problems
    - ✓ Concurrent-access anomalies
    - ✓ Security problems

Data redundancy and inconsistency

This redundancy leads to higher storage and access cost. In addition, it may lead to **data inconsistency**; that is, the various copies of the same data may no longer agree.

Difficulty in accessing data

Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

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**.

Atomicity problems

A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure.

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.

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.

Advantages of DBMS:

- Redundancy can be reduced
- Inconsistency can be avoided
- The data can be shared
- Standards can be enforced
- Security can be enforced
- Integrity can be maintained

Disadvantages of DBMS:

- Confidentiality, Privacy and Security
- Enterprise Vulnerability
- The cost of using a DBMS

DIFFERENCE BETWEEN FILE SYSTEM AND DBMS

Database Systems	Conventional File systems
Data redundancy is less	Data redundancy is more
Security is high	Security is very low
Database systems are used when security constraints are high	Conventional file systems are used where there is less demand for security constraints.
Database systems define the data in a structured manner. Also there is well defined co-relation among the data.	File systems define the data in un-structured manner. Data is usually in isolated form.
Data inconsistency is less in database systems.	Data inconsistency is more in database systems.

1.2 VIEW OF DATA

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

- **Data Abstraction**
- **Instances and Schemas**
- **Data Models**

Data Abstraction

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

- **Physical level**

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

- **Logical level**

The next-higher level of abstraction describes *what* data are stored in the database, and what relationships exist among those data.

Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

- **View level**

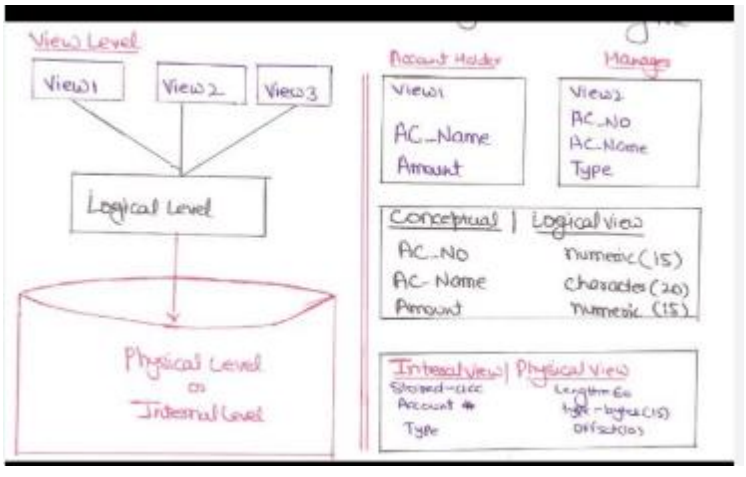
The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Many high-level programming languages support the notion of a structured type. For example, we may describe a record as follows:

type *instructor* = **record**
ID :**char** (5);
name :**char** (20);
dept name :**char** (20);
salary :**numeric** (8,2);
end;

This code defines a new record type called *instructor* with four fields. Each field has a name and a type associated with it.

.



1.2.1 Instances and Schemas

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

Example:

Roll No	Name	Marks

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

Example:

Roll No	Name	Marks
1011	AAA	68
1012	BBB	81

Types of Schema:

Database systems have several schemas, partitioned according to the levels of abstraction.

- The **physical schema** describes the database design at the physical level of abstraction.
- the **logical schema** describes the database design at the logical level of abstraction.
- A database may also have several schemas at the view level, sometimes called **subschemas**, that describe different views of the database.

1.3 DATA MODELS

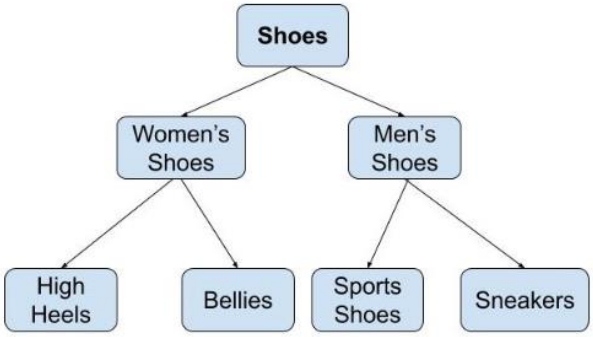
Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

1.3.1 Different types of data model

- Hierarchical model
- Network model
- Entity-relationship model(Graphical/Pictorial representation model)
- Relational model(Record based model)
- Object relational model(Programming model)

1.3.1.1 Hierarchical Model

A hierarchical data model is a data model which the data is organized into a tree like structure. The structure allows repeating information using parent/child relationships:each parent can have many children but each child only has one parent(Fig 1.2). All attributes of a specific record are listed under an entity type.

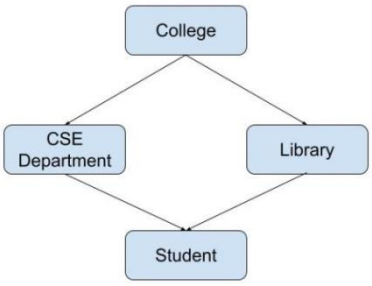


Advantages:

1. The representation of records is done using an ordered tree, which is natural method of implementation of one–to-many relationships.
2. Proper ordering of the tree results in easier and faster retrieval of records.
3. Allows the use of virtual records.

1.3.1.2 Network Model

This model is an extension of the hierarchical model. It was the most popular model before the relational model. This model is the same as the hierarchical model, the only difference is that a record can have more than one parent. It replaces the hierarchical tree with a graph.



Advantages:

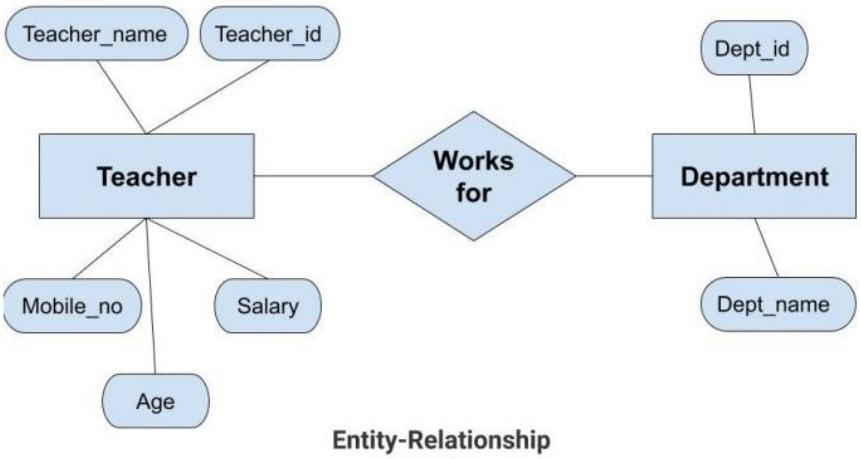
- The data can be accessed faster as compared to the hierarchical model. This is because the data is more related in the network model and there can be more than one path to reach a particular node. So the data can be accessed in many ways.

- As there is a parent-child relationship so data integrity is present. Any change in parent record is reflected in the child record.

1.3.1.3 Entity-Relationship Model

The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities*, and of *relationships* among these objects(Fig 1.1b)

- An **entity** is a "thing" or "object" in the real world that is distinguishable from other objects.
- A **relationship** is an association among several entities.
- Entities are described in a database by a set of **attributes**.



Entity-Relationship Model

1.3.1.4 Entity-Relationship Model

The Relational Model uses a collection of tables both data and the relationship among those data. Each table have multiple columns and each column has a unique name.

Relational database comprising of two tables

Customer –Table.

Customer-Name	Security Number	Address	City	AccountNumber
Preethi	111-222-3456	Yelhanka	Bangalore	A-101
Sharan	111-222-3457	Hebbal	Bangalore	A-125
Preethi	112-123-9878	Jaynagar	Bangalore	A-456
Arun	123-987-9909	MG road	Bangalore	A-987
Preethi	111-222-3456	Yelhanka	Bangalore	A-111
Rocky	222-232-0987	Sanjay Nagar	Bangalore	A-111

Account –Table

Account-Number	Balance
A-101	1000.00
A-125	1200.00
A-456	5000.00
A-987	1234.00
A-111	3000.00

Advantages

1. The main advantage of this model is its ability to represent data in a simplified format.
2. The process of manipulating record is simplified with the use of certain key attributes used to retrieve data.
3. Representation of different types of relationship is possible with this model.

1.3.1.5 Object Relational Model (Programming Model)

As the name suggests it is a combination of both the relational model and the object-oriented model. This model was built to fill the gap between object-oriented model and the relational model. We can have many advanced features like we can make complex data types according to our requirements using the existing data types. The problem with this model is that this can get complex and difficult to handle. So, proper understanding of this model is required.

1.4 DATABASE SYSTEM ARCHITECTURE

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.

1.4.1 Storage Manager

The storage manager translates the various DML statements into low-level file-system commands. The storage manager is responsible for storing, retrieving, and updating data in the database.

- **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 storage manager implements several data structures as part of the physical system implementation(Fig 1.3a):

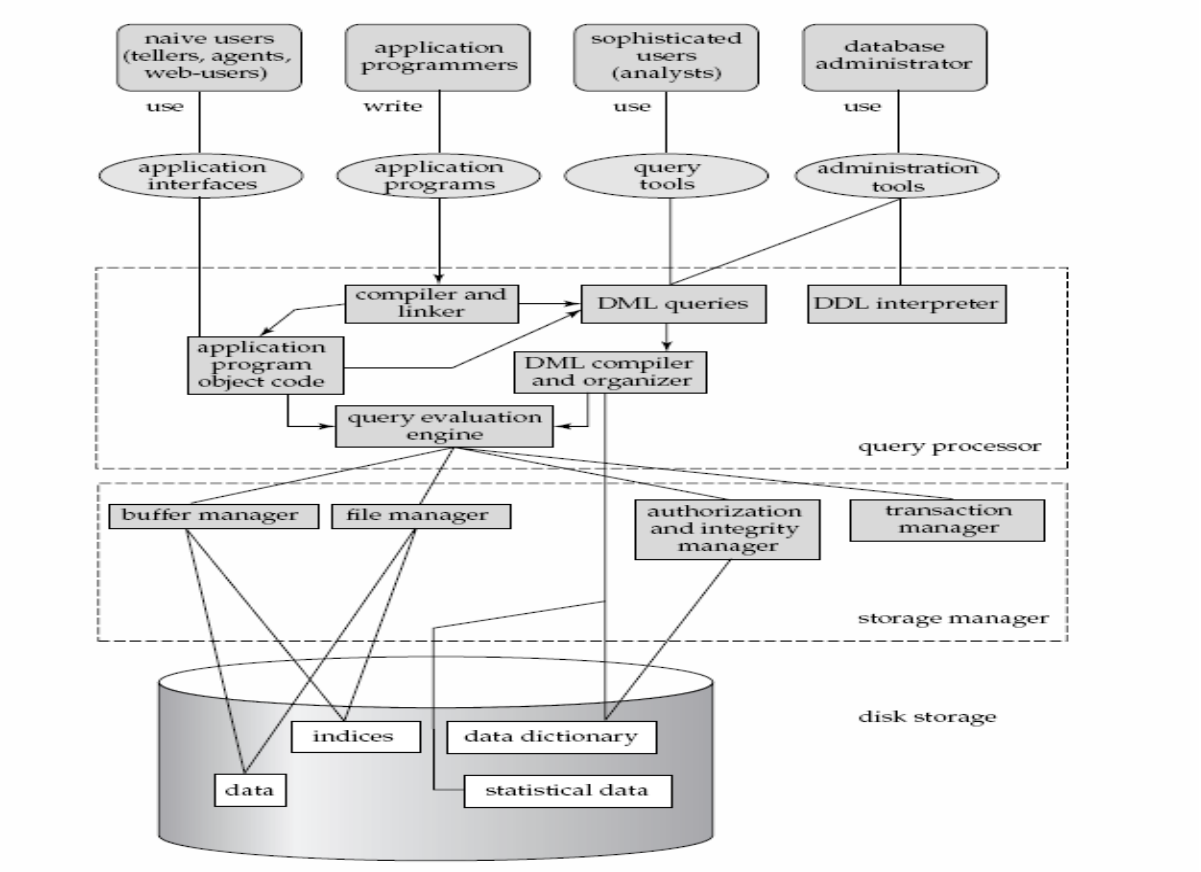


Fig:1.3a Database System Architecture

- **Data files**, which store the database itself.
- **Data dictionary**, which stores metadata about the structure of the database, in particular the schema of the database.
- **Indices**, which provide fast access to data items that hold particular values.

1.4.2 The Query Processor

The query processor components include

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.

- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions
- **Query evaluation engine**, which executes low-level instructions

Database Users:

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

- **Application programmers** – interact with the system through DML calls
- **Sophisticated users** – form requests in a database query language
- **Specialized users** – write specialized database applications that do not fit into the traditional data processing framework
- **Naive users** – invoke one of the permanent application programs that have been written previously

Database applications are usually partitioned into two or three parts, as in Fig 1.3.b

- In a **two-tier architecture**, the application resides at the client machine,where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBCand JDBC are used for interaction between the client and the server.
- In contrast, in a **three-tier architecture**, the client machine acts as merely afront end and does not contain any direct database calls. Instead, the client endcommunicates with an **application server**, usually through a forms interface.The application server in turn communicates with a database system to accessdata.

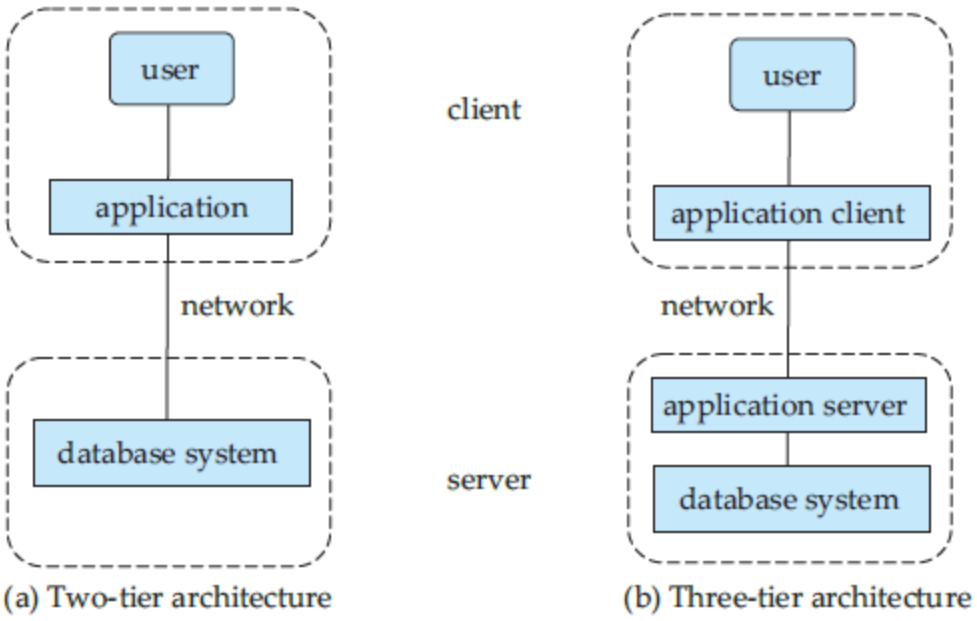


Fig 1.3 b One tier and Two Tier Architectures

1.5 INTRODUCTION TO THE RELATIONAL MODEL

1.5.1 Keys

- We must have a way to specify how tuples within a given relation are distinguished. This is expressed in terms of their attributes.
- In other words, no two tuples in a relation are allowed to have exactly the same value for all attributes.
- 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.
- We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called **candidate keys**.
- A relation, say *r*1, may include among its attributes the primary key of another relation, say *r*2. This attribute is called a **foreign key** from *r*1, referencing *r*2. The relation *r*1 is also called the **referencing relation** of the foreign key dependency, and *r*2 is called the **referenced relation** of the foreign key.

1.5.2 Schema Diagrams

- A database schema, along with primary key and foreign key dependencies, can be depicted by **schema diagrams**. Figure 1.6 shows the schema diagram for our university organization.
- Each relation appears as a box, with the relation name at the top in blue, and the attributes listed inside the box. Primary key attributes are shown underlined.
- Foreign key dependencies appear as arrows from the foreign key attributes of the referencing relation to the primary key of the referenced relation.

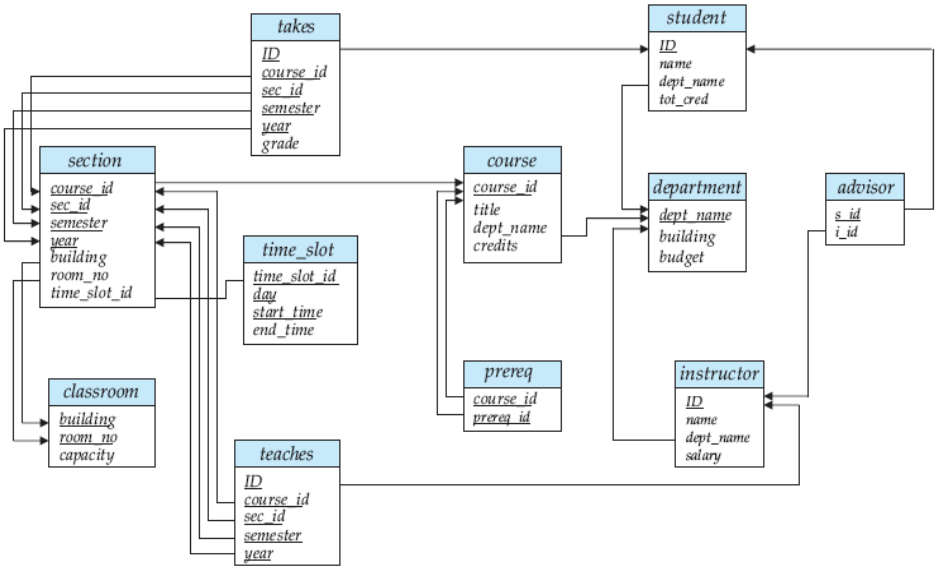


Fig 1.6 Schema diagram for the university database

1.5.3 Relational Query Languages

- A **query language** is a language in which a user requests information from the database. These languages are usually on a level higher than that of a standard programming language.
- Query languages can be categorized as either procedural or nonprocedural.
- In a **procedural language**, the user instructs the system to perform a sequence of operations on the database to compute the desired result.
- In a **nonprocedural language**, the user describes the desired information without giving a specific procedure for obtaining that information.
- There are a number of "pure" query languages: The relational algebra is procedural, whereas the tuple relational calculus and domain relational calculus are nonprocedural.

1.5.4 Relational Operations

- All procedural relational query languages provide a set of operations that can be applied to either a single relation or a pair of relations.
- The *join* operation allows the combining of two relations by merging pairs oftuples, one from each relation, into a single tuple.
- The *Cartesian product* operation combines tuples from two relations, but unlike the join operation, its result contains *all* pairs of tuples from the two relations, regardless of whether their attribute values match.

- The *union* operation performs a set union of two "similarly structured" tables (say a table of all graduate students and a table of all undergraduate students). For example, one can obtain the set of all students in a department. Other set operations, such as *intersection* and *set difference* can be performed as well.

1.6 RELATIONAL ALGEBRA:

Basic operations:

- Selection (σ) Selects a subset of rows from relation.
- Projection (π) Selects a subset of columns from relation.
- Cross-product (×) allows us to combine two relations.
- Set-difference (–) Tuples in relation1, but not in relation2.
- Union (U) Tuples in relation1 and in relation2.
- Rename (ρ) Use new name for the Tables or fields.

Additional operations: Intersection (∩), join (⋈), division (÷): Not essential, but (very!) useful.

Since each operation returns a relation, operations can be composed! (Algebra is "closed".)

Projection: Deletes attributes that are not in projection list.

Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation. (Unary Operation)

Projection operator has to eliminate duplicates! (as it returns a relation which is a set)

Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Duplicate values may be representing different real world entity or relationship)

Consider the BOOK table:

Acc-No	Title	Author
100	"DBMS"	"Silbershatz"
200	"DBMS"	"Ramanuj"
300	"COMPILER"	"Silbershatz"
400	"COMPILER"	"Ullman"
500	"OS"	"Sudarshan"
600	"DBMS"	"Silbershatz"

πTitle(BOOK) =

Title
"DBMS"
"COMPILER"
"OS"

Selection:

Selects rows that satisfy selection condition.

No duplicates in result.

Schema of result identical to schema of (only) input relation.

Result relation can be the input for another relational algebra operation! (Operator composition.)

σAcc-no>300(BOOK) =

Acc-No	Title	Author
400	"COMPILER"	"Ullman"
500	"OS"	"Sudarshan"
600	"DBMS"	"Silbershatz"

σTitle="DBMS"(BOOK)=

Acc-No	Title	Author
100	"DBMS"	"Silbershatz"
200	"DBMS"	"Ramanuj"
600	"DBMS"	"Silbershatz"

πAcc-no (σTitle="DBMS" (BOOK))=

Acc-No
100
200
600

Union, Intersection, Set-Difference

All of these operations take two input relations, which must be union-compatible:

Same number of fields.

`Corresponding' fields have the same type.

What is the schema of result?

Consider:

Borrower

Cust-name	Loan-no
Ram	L-13
Shyam	L-30
Suleman	L-42

Depositor

Cust-name	Acc-no
Suleman	A-100
Radheshyam	A-300
Ram	A-401

List of customers who are either borrower or depositor at bank= πCust-name (Borrower) U πCust-name (Depositor) =

Cust-name
Ram
Shyam
Suleman
Radeshyam

Customers who are both borrowers and depositors = πCust-name (Borrower) ∩ πCust-name (Depositor) =

Cust-name
Ram
Suleman

Customers who are borrowers but not depositors = πCust-name (Borrower) – πCust-name (Depositor) =

Cust-name
Shyam

Cartesian-Product or Cross-Product (S1 × R1)

Each row of S1 is paired with each row of R1.

Result schema has one field per field of S1 and R1, with field names `inherited' if possible. Consider the borrower and loan tables as follows:

Borrower:

Cust-name	Loan-no
Ram	L-13
Shyam	L-30
Suleman	L-42

Loan:

Loan-no	Amount
L-13	1000
L-30	20000
L-42	40000

Cross product of Borrower and Loan, Borrower × Loan =

Borrower.Cust-name	Borrower.Loan-no	Loan. Loan-no	Loan. Amount
Ram	L-13	L-13	1000
Ram	L-13	L-30	20000
Ram	L-13	L-42	40000
Shyam	L-30	L-13	1000
Shyam	L-30	L-30	20000
Shyam	L-30	L-42	40000
Suleman	L-42	L-13	1000
Suleman	L-42	L-30	20000
Suleman	L-42	L-42	40000

The rename operation can be used to rename the fields to avoid confusion when two field names are same in two participating tables:

For example the statement, ρLoan-borrower(Cust-name,Loan-No-1, Loan-No-2,Amount)(Borrower × Loan) results into- A new Table named Loan-borrower is created where it has four fields which are renamed as Cust-name, Loan-No-1, Loan-No-2 and Amount and the rows contains the same data as the cross product of Borrower and Loan.

Loan-borrower:

Cust-name	Loan-No-1	Loan-No-2	Amount
Ram	L-13	L-13	1000
Ram	L-13	L-30	20000
Ram	L-13	L-42	40000
Shyam	L-30	L-13	1000
Shyam	L-30	L-30	20000
Shyam	L-30	L-42	40000
Suleman	L-42	L-13	1000
Suleman	L-42	L-30	20000
Suleman	L-42	L-42	40000

Rename Operation:

It can be used in two ways:

$\rho_x(E)$ return the result of expression E in the table named x.

$\rho_{x(A_1,A_2,\ldots,A_n)}(E)$ return the result of expression E in the table named x with the attributes renamed to A1, A2,…, An.

Its benefit can be understood by the solution of the query "Find the largest account balance in the bank"

It can be solved by following steps:

Find out the relation of those balances which are not largest.

Consider Cartesian product of Account with itself i.e. Account × Account

Compare the balances of first Account table with balances of second Account table in the product.

For that we should rename one of the account tables by some other name to avoid the confusion

It can be done by following operation

ΠAccount.balance (σAccount.balance<d.balance(Account× ρd(Account))

So the above relation contains the balances which are not largest.

Subtract this relation from the relation containing all the balances i.e .Πbalance (Account).

So the final statement for solving above query is

Πbalance (Account) - ΠAccount.balance (σAccount.balance<d.balance(Account× ρd(Account))

Additional Operations

Natural Join ($S_1 \bowtie R_1$)

Forms Cartesian product of its two arguments, performs selection forcing equality on those attributes that appear in both relations

For example consider Borrower and Loan relations, the natural join between them Borrower ⋈ Loan will automatically perform the selection on the table returned by Borrower × Loan which force equality on the attribute that appear in both Borrower and Loan i.e. Loan-no and also will have only one of the column named Loan-No.

That means

Borrower ⋈ Loan = σBorrower.Loan-no = Loan.Loan-no (Borrower × Loan).

The table returned from this will be as follows:

Eliminate rows that does not satisfy the selection criteria "σBorrower.Loan-no = Loan.Loan-no" from Borrower × Loan =

Borrower. Cust-name	Borrower. Loan-no	Loan. Loan-no	Loan. Amount
Ram	L-13	L-13	1000
Ram	L-13	L-30	20000
Ram	L-13	L-42	40000
Shyam	L-30	L-13	1000
Shyam	L-30	L-30	20000
Shyam	L-30	L-42	40000
Suleman	L-42	L-13	1000
Suleman	L-42	L-30	20000
Suleman	L-42	L-42	40000

And will remove one of the columns named Loan-no.

i.e.Borrower ⋈ Loan =

Cust-name	Loan-no	Amount
Ram	L-13	1000
Shyam	L-30	20000
Suleman	L-42	40000

Division Operation:

Division Operation is represented by "division"(÷ or /) operator and is used in queries that involve keywords "every", "all", etc.

Notation : R(X,Y)/S(Y)

Here,

R is the first relation from which data is retrieved.

S is the second relation that will help to retrieve the data.

X and Y are the attributes/columns present in relation. We can have multiple attributes in relation, but keep in mind that attributes of S must be a proper subset of attributes of R.

For each corresponding value of Y, the above notation will return us the value of X from tuple<X,Y> which exists everywhere.

Eg:

ENROLLED

STUDENT_ID	COURSE_ID
Student_1	DBMS
Student_2	DBMS
Student_1	OS
Student_3	OS

COURSE

COURSE_ID
DBMS
OS

ENROLLED(STUDENT_ID, COURSE_ID)/COURSE(COURSE_ID)

Output:

STUDENT_ID
Student_1