UNIT-1

Introduction to Database systems

DATABASE:-A database is a collection of information that is organized so that it can be easily accessed, managed and updated. Data is organized into rows, columns and tables, and it is indexed to make it easier to find relevant information. Data gets updated, expanded and deleted as new information is added. Databases process workloads to create and update themselves, querying the data they contain and running applications against it. **DATA: -** Any factor that can be stored.
Example: text, numbers, images, videos and speech.

Database Applications: A Database application is a computer program whose primary purpose is entering and retrieving information from a computerized database.

Banking: all transactions Airlines:
reservations, schedules Universities:
registration, grades Sales: customers,
products, purchases
Online retailers: order tracking, customized recommendations
Manufacturing: production, inventory, orders, supply chain
Human resources: employee records, salaries, tax deductions
Databases touch all aspects of our lives

What Is a DBMS?

A *Database Management System (DBMS)* is a software package designed to interact with end-users, other applications, store and manage databases. A general-purpose DBMS allows the definition, creation, querying, update, and administration of databases.

DBMS contains information about a particular enterprise

Collection of interrelated data

Set of programs to access the data

An environment that is both *convenient* and *efficient* to use

Why Use a DBMS?

A database management system stores, organizes and manages a large amount of information within a single software application. It manages data efficiently and allows users to perform multiple tasks with ease.

Reduced application development time.

Data integrity and security.

Uniform data administration.

Concurrent access, recovery from crashes.

Why Study Databases??

Shift from *computation* to *information* at the "low end": scramble to webspace (a mess!) at the "high end": scientific applications

Datasets increasing in diversity and volume. Digital libraries, interactive video, Human Genome project, EOS project ... need for DBMS exploding

DBMS encompasses most of CS OS, languages, theory, AI, multimedia, logic.

Purpose of Database Systems:

In the early days, database applications were built directly on top of file systems. A DBMS provides users with a systematic way to create, retrieve, update and manage data. It is a middleware between the databases which store all the data and the users or applications which need to interact with that stored database. A DBMS can limit what data the end user sees, as well as how that end user can view the data, providing many views of a single database schema.

Database + database management system = database system

Drawbacks of using file systems to store data:

Data redundancy and inconsistency.

Multiple file formats, duplication of information in different files.

Difficulty in accessing data.

Need to write a new program to carry out each new task.

Data isolation — multiple files and formats

Integrity problems

Hard to add new constraints or change existing ones

Atomicity of updates

Failures may leave database in an inconsistent state with partial updates carried out

Example: Transfer of funds from one account to another should either complete or not happen at all

Concurrent access by multiple users

Concurrent accessed needed for performance

Uncontrolled concurrent accesses can lead to inconsistencies

Example: Two people reading a balance and updating it at the same time

Security problems

Hard to provide user access to some, but not all, data

Database systems offer solutions to all the above problems

Files vs. DBMS:

A file processing system is a collection of programs that store and manage files in computer hard-disk. On the other hand, a database management system is collection of programs that enables to create and maintain a database. File processing system has more data redundancy, less data redundancy in dbms.

Application must stage large datasets between main memory and secondary

Storage (e.g., buffering, page-oriented access, 32-bit addressing, etc.)

Special code for different queries

Must protect data from inconsistency due to multiple concurrent users

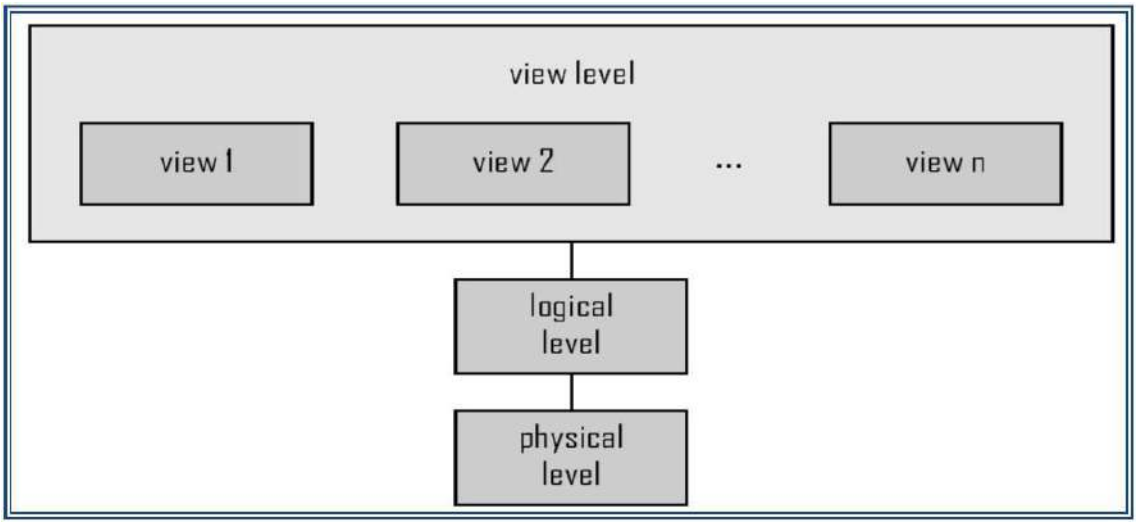
Crash recovery

Security and access control

View of Data

Architecture for a database system:

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. The main task of database system is to provide abstract view of data i.e hides certain details of storage to the users.



Data Abstraction:

Major purpose of dbms is to provide users with abstract view of data i.e. the system hides cert -ain details of how the data are stored and maintained. Since database system users are not computer trained, developers hide the complexity from users through 3 levels of abstraction, to simplify user's interaction with the system.

Levels of Abstraction

Physical level of data abstraction: Describes how a record (e.g., customer) is stored. This is the lowest level of abstraction which describes how data are actually stored.

Logical level of data abstraction: The next highest level of abstraction which hides what data are actually stored in the database and what relations hip exists among them. Describes data stored in database, and the relationships among the data.

```
type customer = record;
customer_id:string;
customer_name:string;
```

customer_stree:string;

customer city : string;

end;

View Level of data abstraction: The highest level of abstraction provides security mechanism to prevent user from accessing certain parts of database. Application programs hide details of data types. Views can also hide information (such as an employee's salary) for security purposes and to simplify the interaction with the system.

Summary

DBMS used to maintain, query large datasets.

Benefits include recovery from system crashes, concurrent access, quick application development, data integrity and security.

Levels of abstraction give data independence.

A DBMS typically has a layered architecture.

DBAs hold responsible jobs and are well-paid!

DBMS R&D is one of the broadest, most exciting areas in CS.

Instances and Schemas:

Similar to types and variables in programming languages. Database changes over time when information is inserted or deleted.

Instance – the actual content of the database at a particular point in time analogous to the value of a variable is called an instance of the database.

Schema – the logical structure of the database called the database schema. Schema is of three types: Physical schema, logical schema and view schema.

Example: The database consists of information about a set of customers and accounts and the relationship between them) Analogous to type information of a variable in a program

Physical schema: Database design at the physical level is called physical schema. How the data stored in blocks of storage is described at this level.

Logical schema: database design at the logical level Instances and schemas, programmers and database administrators work at this level, at this level data can be described as certain types of data records gets stored in data structures, however the internal details such as implementation of data structure is hidden at this level.

View schema: Design of database at view level is called view schema**.** This generally describes end user interaction with database systems.

Physical Data Independence – The ability to modify the physical schema without changing the logical schema.

Applications depend on the logical schema

In general, the interfaces between the various levels and components should be well defined so

that changes in some parts do not seriously influence others.

Example: University Database

Conceptual schema:

Students(sid: string, name: string, login: string, age: integer, gpa:real)

Courses(cid: string, cname:string, credits:integer)

Enrolled(sid:string, cid:string, grade:string)

Physical schema: Relations stored as unordered files.

Index on first column of Students.

External Schema (View):

Course_info(cid:string,enrollment:integer)

Data Independence:

Applications insulated from how data is structured and stored.

Logical data independence: Protection from changes in *logical* structure of data.

Physical data independence: Protection from changes in *physical* structure of data.

History of Database Systems:

1950s and early 1960s:

–Data processing using magnetic tapes for storage

Tapes provide only sequential access

–Punched cards for input

Late 1960s and 1970s:

–Hard disks allow direct access to data

–Network and hierarchical data models in widespread use

–Ted Codd defines the relational data model

Would win the ACM Turing Award for this work

IBM Research begins System R prototype

UC Berkeley begins Ingres prototype

–High-performance (for the era) transaction processing

1980s:

–Research relational prototypes evolve into commercial systems

SQL becomes industry standard

–Parallel and distributed database systems

–Object-oriented database systems

1990s:

–Large decision support and data-mining applications

–Large multi-terabyte data warehouses

–Emergence of Web commerce

2000s:

–XML and XQuery standards

–Automated database administration

–Increasing use of highly parallel database systems

–Web-scale distributed data storage systems

Data Models:

A Data Model is a logical structure of Database. It is a collection of concepts for describing data, reflects entities, attributes, relationship among data, constrains etc. A schema is a description of a particular collection of data, using the given data model. The relational model of data is the most widely used model today. it is a collection of tools for describing

- Data
- Data relationships
- Data semantics
- Data constraints
- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi structured data model (XML)
- Other older models:
  Network model
  Hierarchical model

Every relation has a schema, which describes the columns, or fields.

Different types of data models are:

Relational model: The relational model uses a collection of tables to represent both data and relationships among those data. Each table has multiple columns with unique name.

- It is example of record based model

- These models are structured is fixed-format of several types.
- Each table contains records of particular type
- Each record type defines fixed number of fields, or attributes.
- The columns of the table correspond to attributes of the record type.

The relational data model is the most widely used data model and majority of current database systems are based on relational model.

Entity-relationship model: The E-R model is based on a perception of real world that consists of basic objects called entities and relationships among these objects. An entity is a 'thing' or 'object' in the real world, E-R model is widely used in database design.

Introduction to Database Design:

Conceptual design: *(ER Model is used at this stage.)*

–What are the *entities* and *relationships* in the enterprise?

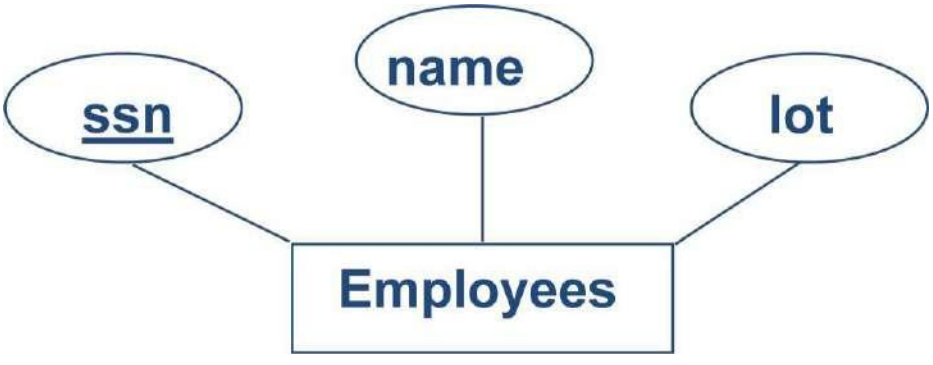
–What information about these entities and relationships should we store in the database?

–What are the *integrity constraints* or *business rules* that hold?

–A database `schema' in the ER Model can be represented pictorially (*ER diagrams*).

–Can map an ER diagram into a relational schema.

ER Model:



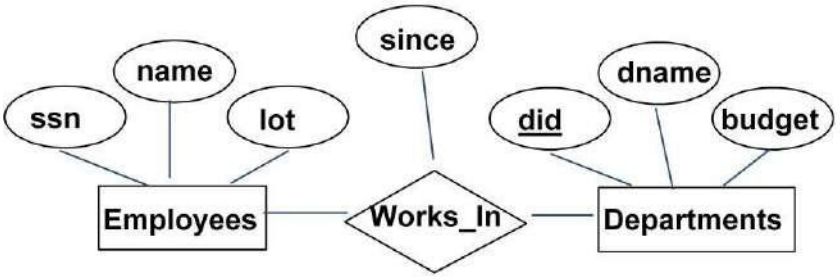
Entity: Real-world objects distinguishable from other objects. An entity is described (in DB) using a set of *attributes*.

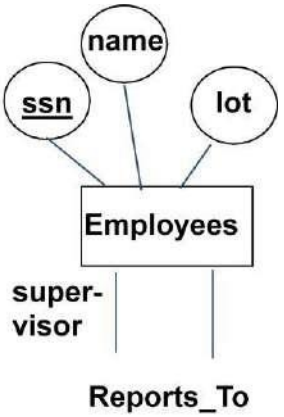
Entity Set: A collection of similar entities. E.g., all employees.

– All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)

– Each entity set has a *key*.

– Each attribute has a *domain*.





Relationship: Association among two or more entities. E.g., Attishoo works in Pharmacy department.

Relationship Set: Collection of similar relationships.

–An n-ary relationship set R relates n entity sets E1 ... En; each relationship in R involves entities e1 E1, ..., en En

Same entity set could participate in different relationship sets, or in different "roles" in same set.

Modeling:

A *database* can be modeled as:

–a collection of entities,

–relationship among entities.

Entities and Entity Sets:

An **entity** is an object that exists and is distinguishable from other objects.

Example: specific person, company, event, plant

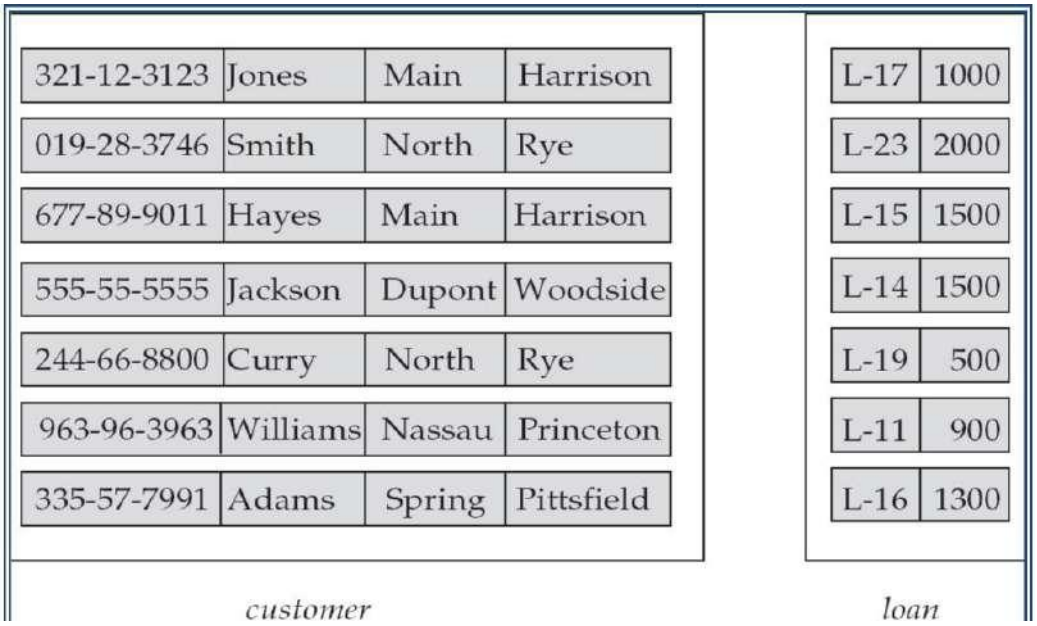
Entities have *attributes*

Example: people have *names* and *addresses*

An **entity set** is a set of entities of the same type that share the same properties.

Example: set of all persons, companies, trees, holidays

Example:Entity Sets customer and loan



321-12-3123	Jones	Main	Harrison
019-28-3746	Smith	North	Rye
677-89-9011	Hayes	Main	Harrison
555-55-5555	Jackson	Dupont	Woodside
244-66-8800	Curry	North	Rye
963-96-3963	Williams	Nassau	Princeton
335-57-7991	Adams	Spring	Pittsfield

customer

L-17	1000
L-23	2000
L-15	1500
L-14	1500
L-19	500
L-11	900
L-16	1300

loan

Attributes:

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

Domain – the set of permitted values for each attribute

Attribute types:

–*Simple* and *composite* attributes.

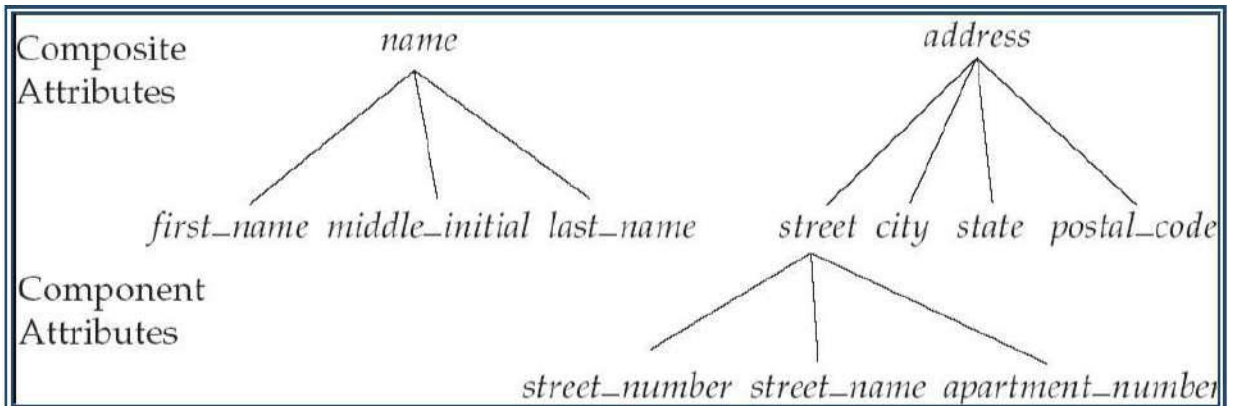
–*Single-valued* and *multi-valued* attributes Example:

multivalued attribute: *phone_numbers*

–*Derived* attributes can be computed from other attributes

Example: age, given date_of_birth

Composite Attributes



Mapping Cardinality Constraints

Express the number of entities to which another entity can be associated via a relationship set.

Most useful in describing binary relationship sets.

For a binary relationship set the mapping cardinality must be one of the following types:

–One to one

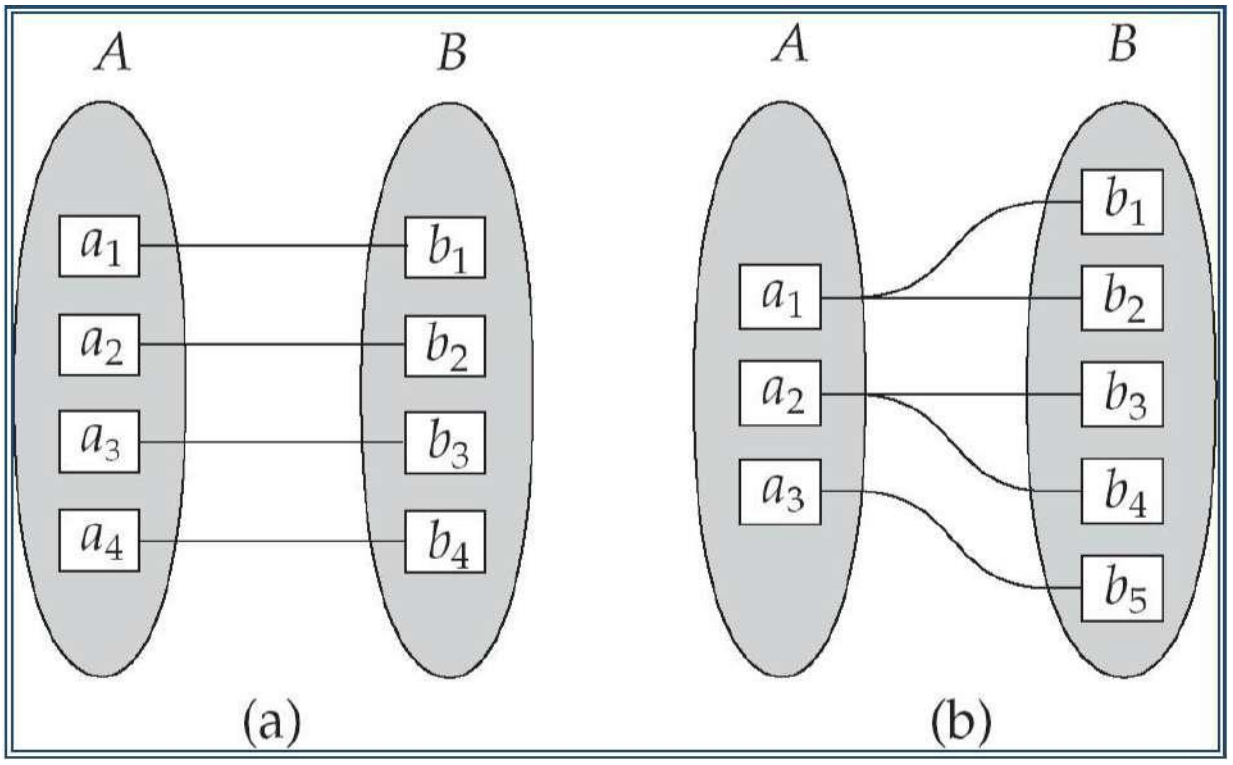
–One to many

–Many to one

–Many to many

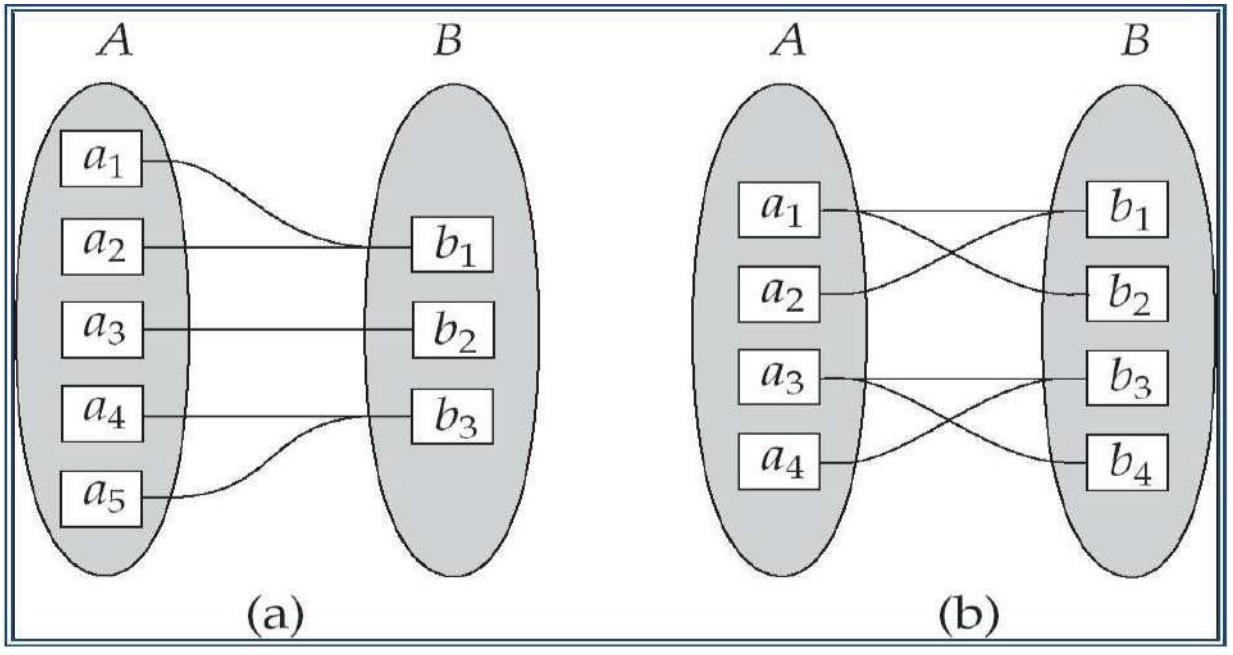
Mapping Cardinalities:

Note: Some elements in *A* and *B* may not be mapped to any elements in the other set



Mapping Cardinalities

Note: Some elements in A and B may not be mapped to any elements in the other set



Relationships and Relationship Sets

A **relationship** is an association among several entities

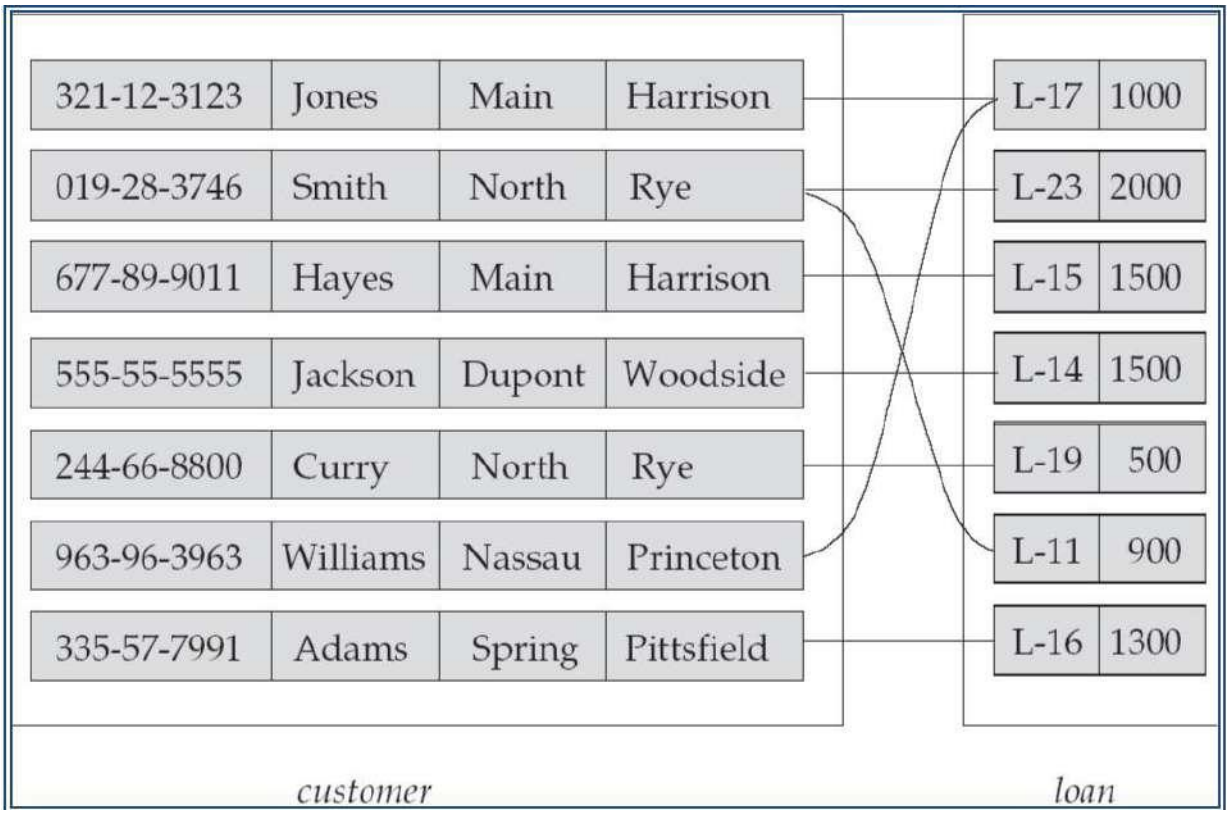
A **relationship set** is a mathematical relation among $n \geq 2$ entities, each taken from entity sets

$\{(e_1, e_2, \ldots e_n) \mid e_1 \in E_1, e_2 \in E_2, \ldots, e_n \in E_n\}$ where $(e_1, e_2, \ldots, e_n)$ is a relationship

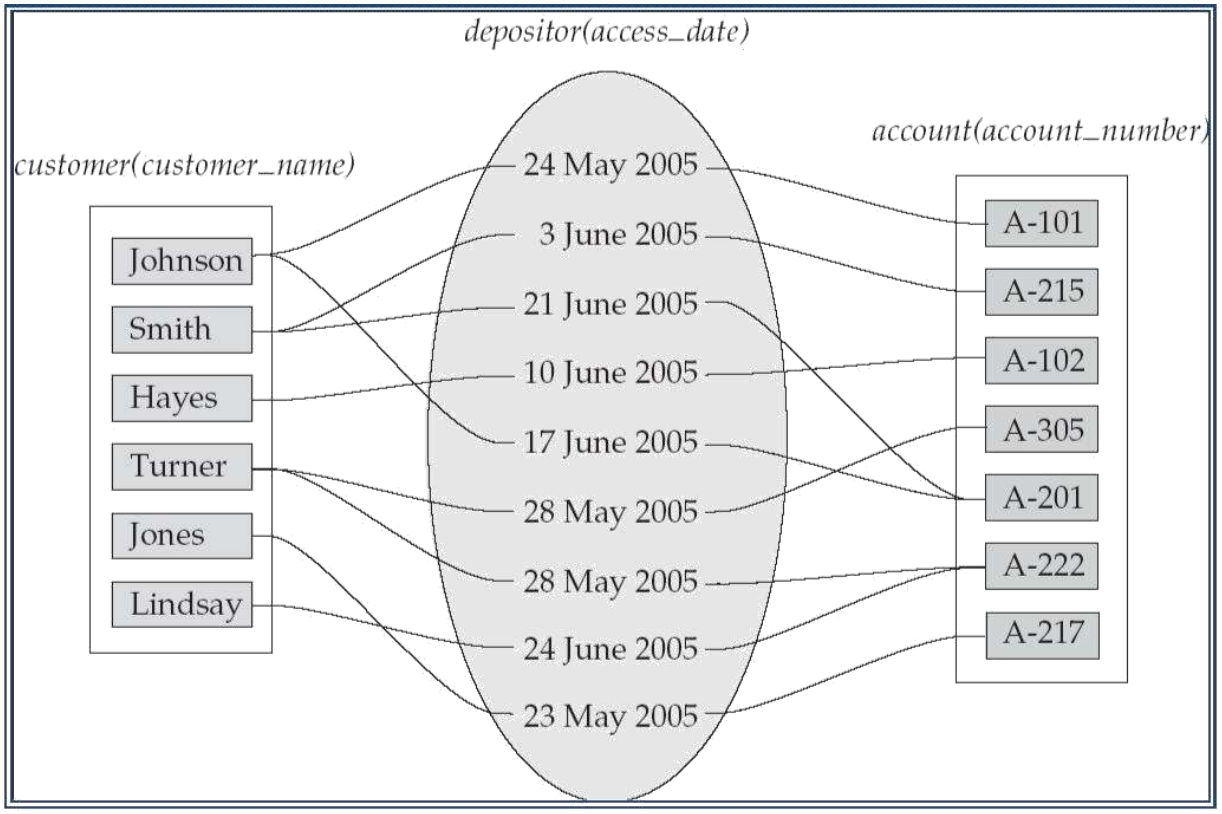
– Example:

(Hayes, A-102) $\in$ *depositor*

Relationship Set *borrower*



An **attribute** can also be property of a relationship set.



For instance, the *depositor* relationship set between entity sets *customer* and *account*

may have the attribute *access-date*

Degree of a Relationship Set

Refers to number of entity sets that participate in a relationship set.

Relationship sets that involve two entity sets are **binary** (or degree two). Generally, most relationship sets in a database system are binary.

Relationship sets may involve more than two entity sets.

Example: Suppose employees of a bank may have jobs (responsibilities) at multiple branches, with different jobs at different branches. Then there is a ternary relationship set between entity sets *employee, job, and branch*

Relationships between more than two entity sets are rare. Most relationships are binary.

Weak Entities

A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.

- Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
- Weak entity set must have total participation in this *identifying* relationship set.

Weak Entity Sets

An entity set that does not have a primary key is referred to as a **weak entity set**.

The existence of a weak entity set depends on the existence of a **identifying entity set**

- it must relate to the identifying entity set via a total, one-to-many relationship set from the identifying to the weak entity set
- Identifying relationship depicted using a double diamond

The **discriminator** (*or partial key)* of a weak entity set is the set of attributes that distinguishes among all the entities of a weak entity set.

The primary key of a weak entity set is formed by the primary key of the strong entity set on which the weak entity set is existence dependent, plus the weak entity set's discriminator.

depict a weak entity set by double rectangles.

underline the discriminator of a weak entity set with a dashed line.

payment_number – discriminator of the *payment* entity set

Primary key for *payment* – (*loan_number, payment_number*)

Note: the primary key of the strong entity set is not explicitly stored with the weak entity set, since it is implicit in the identifying relationship.

If *loan_number* were explicitly stored, *payment* could be made a strong entity, but then

the relationship between *payment* and *loan* would be duplicated by an implicit relationship defined by the attribute *loan_number* common to *payment* and *loan*

More Weak Entity Set Examples

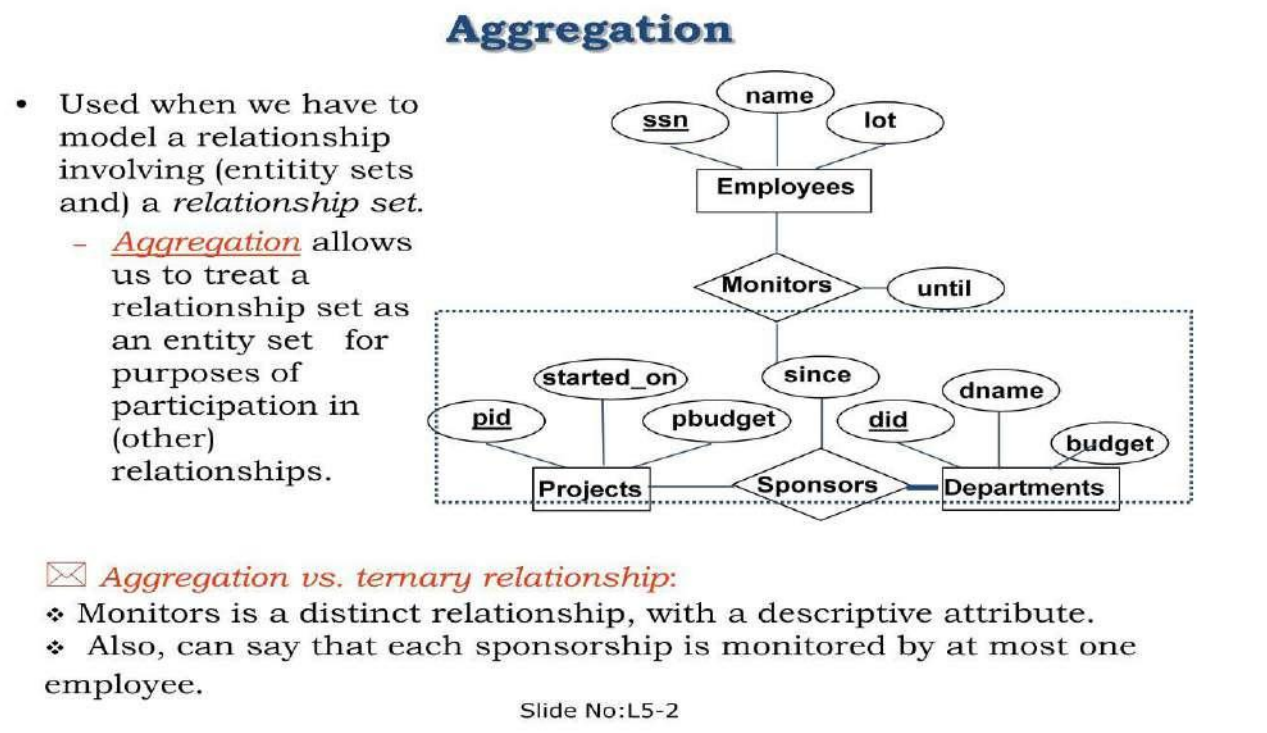
In a university, a *course* is a strong entity and a *course_offering* can be modeled as a weak entity

The discriminator of *course_offering* would be *semester* (including year) and *section_number* (if there is more than one section)

If we model *course_offering* as a strong entity we would model *course_number* as an attribute.

Then the relationship with *course* would be implicit in the *course_number* attribute

Aggregation



Relationship sets *works_on* and *manages* represent overlapping information

– Every *manages* relationship corresponds to a *works_on* relationship

However, some *works_on* relationships may not correspond to any *manages* relationships. So we can't discard the *works_on* relationship

Eliminate this redundancy via *aggregation*

– Treat relationship as an abstract entity

– Allows relationships between relationships

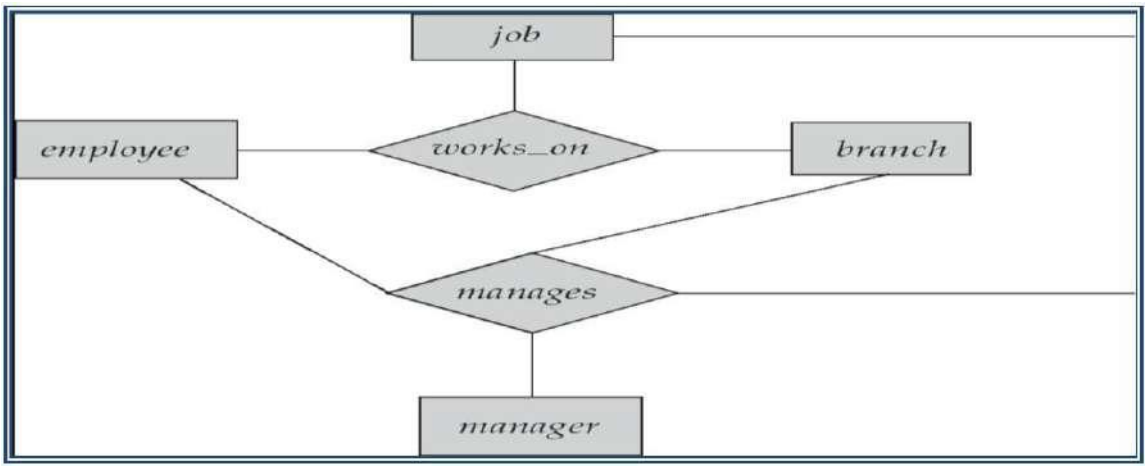
– Abstraction of relationship into new entity

Without introducing redundancy, the following diagram represents:

– An employee works on a particular job at a particular branch

– An employee, branch, job combination may have an associated manager

E-R Diagram with Aggregation



Conceptual Design with ER Model

Design choices:

– Should a concept be modeled as an entity or an attribute?

– Should a concept be modeled as an entity or a relationship?

– Identifying relationships: Binary or ternary? Aggregation?

Constraints in the ER Model:

– A lot of data semantics can (and should) be captured.

– But some constraints cannot be captured in ER diagrams.

Entity vs. Attribute

Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?

Depends upon the use we want to make of address information, and the semantics of the data:

If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).

If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

An example in the other direction: a ternary relation Contracts relates entity sets Parts, Departments and Suppliers, and has descriptive attribute *qty*. No combination of binary . relationships is an adequate substitute:
S "can-supply" P, D "needs" P,and D "deals-with" S does not imply that D has agreed to buy P from S.
–How do we record *qty*?

Introduction to relational model

Relational Database: Definitions

Relational database: a set of *relations*

Relation: made up of 2 parts:

– *Instance* : a *table*, with rows and columns.

#Rows = *cardinality*, #fields = *degree* / *arity*.

– *Schema* : specifies name of relation, plus name and type of each column. E.G.

Students (*sid*: string, *name*: string, *login*: string, *age*: integer, *gpa*: real).

Can think of a relation as a *set* of rows or *tuples* (i.e., all rows are distinct).

Example Instance of Students Relation

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Cardinality = 3, degree = 5, all rows distinct

Do all columns in a relation instance have to be distinct?

Relational Query Languages A major strength of the relational model: supports simple, powerful *querying* of data.

Queries can be written intuitively, and the DBMS is responsible for efficient evaluation.

– The key: precise semantics for relational queries.

– Allows the optimizer to extensively re-order operations, and still ensure that the answer does not change.

Creating Relations in SQL

Creates the Students relation. Observe that the type of each field is specified, and enforced by the DBMS whenever tuples are added or modified.

CREATE TABLE Students (sid CHAR(20), name CHAR(20),login CHAR(10),age: INTEGER, gpa: REAL)

As another example, the Enrolled table holds information about courses that students take.

CREATE TABLE Enrolled (sid: CHAR(20),cid: CHAR(20), grade: CHAR(2))

Integrity Constraints (ICs) over Relations:

IC: condition that must be true for *any* instance of the database; e.g., *domain constraints.*

ICs are specified when schema is defined.

ICs are checked when relations are modified.

A *legal* instance of a relation is one that satisfies all specified ICs.

DBMS should not allow illegal instances.

If the DBMS checks ICs, stored data is more faithful to real-world meaning.

– Avoids data entry errors, too!

Primary Key Constraints

A set of fields is a *key* for a relation if :

No two distinct tuples can have same values in all key fields, and

This is not true for any subset of the key.

– Part 2 false? A *superkey*.

– If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*.

E.g., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.

Primary and Candidate Keys in SQL

Possibly many *candidate keys* (specified using UNIQUE), one of which is chosen as the *primary key*.

Foreign Keys, Referential Integrity

Foreign key : Set of fields in one relation that is used to `refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer'.

E.g. *sid* is a foreign key referring to Students:

Foreign Keys in SQL

Only students listed in the Students relation should be allowed to enroll for courses.

Enforcing Integrity constraints

Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.

What should be done if an Enrolled tuple with a non-existent student id is inserted? (*Reject it!*)

What should be done if a Students tuple is deleted?

- Also delete all Enrolled tuples that refer to it.
- Disallow deletion of a Students tuple that is referred to.
- Set sid in Enrolled tuples that refer to it to a *default sid*.
- (In SQL, also: Set sid in Enrolled tuples that refer to it to a special value *null*, denoting *`unknown'* or *`inapplicable'*.)

Similar if primary key of Students tuple is updated.

Referential Integrity in SQL

SQL/92 and SQL:1999 support all 4 options on deletes and updates.

- Default is NO ACTION (*delete/update is rejected*)
- CASCADE (also delete all tuples that refer to deleted tuple)
- SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

Where do ICs Come From?

ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.

We can check a database instance to see if an IC is violated, but we can NEVER infer

that an IC is true by looking at an instance.

– An IC is a statement about *all possible* instances!

– From example, we know *name* is not a key, but the assertion that *sid* is a key is given to us.

Key and foreign key ICs are the most common; more general ICs supported too.

Data base Languages:

Data Control Language (DCL): It is used to control privilege in database. To perform any operations like creating tables, view and modifying we need privileges which are of two types.

System:- Creating session and tables are types of system privilege.

Object:- Any command or query to work on tables comes under object privilege.

DCL defines two commands GRANT and REVOKE.

GRANT:-Gives user access privilege to database.

REVOKE: - To take back permissions from users.

CONNECTING TO ORACLE:

CONNECT<USER NAME>/<PASSWORD>@<DATABASE NAME>;

Create user login:

CREATE USER <USER_NAME> IDENTIFIED BY <PASSWORD>;

Provide roles:

GRANT CONNECT, CREATE SESSION, RESOURCE TO <USER_NAME>;

Provide privileges:

GRANT ALL PRIVILEGES TO <USER_NAME>;

Data Definition Language (DDL):

Specification notation for defining the database schema by a set of definitions.

DDL compiler generates a set of tables stored in a *data dictionary*

Data dictionary contains metadata (i.e., data about data)

Database schema

Data *storage and definition* language

Specifies the storage structure and access methods used

Integrity constraints

Domain constraints

Referential integrity (e.g. *branch_name* must correspond to a valid to branch in the *branch* table)

Authorization

Procedural – user specifies what data is required and how to get those data

Declarative (nonprocedural) – user specifies what data is required without specifying how to get those data.

DDL: Data Definition Language

All DDL commands are auto-committed. That means it saves all the changes permanently in the database.

Command	Description
create	to create new table or database
alter	for alteration
truncate	delete data from table
drop	to drop a table
rename	to rename a table

CREATE command:

create is a DDL command used to create a table or a database.

Creating a database

To create a database in RDBMS, *create* command is uses. Following is the Syntax,

Create database database-name;

Example for creating database

Create database *Test*;

The above command will create a database named **Test**.

Creating a table

create command is also used to create a table. We can specify names and datatypes of various columns along. Following is the Syntax,

create table *table-name*

{

Column-name1 *datatype1*,

Column-name2 *datatype2*,

Column-name1 *datatype3*,

Column-name2 *datatype4*

};

Create table command will tell the database system to create a new table with given table name and column information.

Example for creating table

Create table *Student*(id *int*, name *varchar*, age *int*);

The above command will create a new table **Student** in database system with 3 columns, namely id, name and age.

ALTER command

alter command is used for alteration of table structures. There are various uses of *alter* command, such as,

to add a column to existing table

to rename any existing column

to change datatype of any column or to modify its size.

Alter is also used to drop a column.

To add column to existing table

Using alter command we can add a column to an existing table. Following is the Syntax,

Alter table *table-name* add(column-name *datatype*);

Here is an Example for this,

Alter table *student* add(address *char*);

The above command will add a new column *address* to the **Student** table

To add multiple column to existing table

Using alter command we can even add multiple columns to an existing table. Following is the Syntax,

Alter table *table-name* add(column1 *datatype1*, column2 *datatype2*, column3 *datatype3*, colum4 *datatype4*);

Here is an Example for this,

Alter table *student* add(father_name *varchar(60)*, mother_name v*archar(60)*, DOB *date*);

Date input format is:- 'date-month-year' i.e '10-jan-2016'

The above command will add three new columns to the **Student** table

To add column with default value

alter command can add a new column to an existing table with default values. Following is the Syntax,

alter table table_name add (column_name *datatype* default *data*);

Here is an Example for this,

alter table Student add(branch *char* default 'CSE');

The above command will add a new column with default value to the **Student** table

---

To modify an existing column

alter command is used to modify data type of an existing column . Following is the Syntax,

alter table *table-name* modify(column-name *datatype*);

Here is an Example for this,

alter table *Student* modify(address *varchar(30)*);

The above command will modify *address* column of the **Student table**

To rename a column

Using alter command you can rename an existing column. Following is the Syntax,

alter table *table-name* rename *old-column-name* to new-*column-name*;

Here is an Example for this,

alter table *Student* rename address to Location;

The above command will rename *address* column to *Location*.

To drop a column

alter command is also used to drop columns also. Following is the Syntax,

alter table *table-name* drop(column-name);

Here is an Example for this,

alter table *Student* drop(address);

The above command will drop *address* column from the **Student table**

SQL queries to Truncate, Drop or Rename a Table

truncate command

truncate command removes all records from a table. But this command will not destroy the table's structure. When we apply truncate command on a table its Primary key is initialized. Following is its Syntax,

truncate table *table-name*

Here is an Example explaining it.

truncate table *Student*;

The above query will delete all the records of **Student** table.

truncate command is different from **delete** command. delete command will delete all the rows from a table whereas truncate command re-initializes a table(like a newly created table).

eg. If you have a table with 10 rows and an auto_increment primary key, if you use *delete* command to delete all the rows, it will delete all the rows, but will not initialize the primary key, hence if you will insert any row after using delete command, the auto_increment primary key will start from 11. But in case of *truncate*command, primary key is re-initialized.

drop command

drop query completely removes a table from database. This command will also destroy the table structure. Following is its Syntax,

drop table *table-name;*

Here is an Example explaining it.

drop table *Student*;

The above query will delete the **Student** table completely. It can also be used on Databases. For Example, to drop a database,

drop database *Test*;

The above query will drop a database named **Test** from the system.

rename query

rename command is used to rename a table. Following is its Syntax,

rename table *old-table-name* to *new-table-name*

Here is an Example explaining it.

rename table *Student* to *Student-record*;

The above query will rename **Student** table to **Student-record**.

DML COMMANDS:

INSERT command

Insert command is used to insert data into a table. Following is its general syntax,

insert into *table_name* values(data1,data2,…….);

Lets see an example,

Consider a table **Student** with following fields.

S_id	S_Name	age

INSERT into *Student* values(101,'Adam',15);

The above command will insert a record into **Student** table.

S_id	S_Name	age
101	Adam	15

Example to Insert NULL value to a column

Both the statements below will insert NULL value into **age** column of the Student table.

INSERT into *Student*(id,name) values(102,'Alex');

Or,

INSERT into *Student* values(102,'Alex',null);

The above command will insert only two column value other column is set to null.

S_id	S_Name	age
101	Adam	15
102	Alex	

Example to Insert Default value to a column

INSERT into *Student* values(103,'Chris',default);

S_id	S_Name	age
101	Adam	15
102	Alex	
103	chris	14

Suppose the **age** column of student table has default value of 14.

Also, if you run the below query, it will insert default value into the age column, whatever the default value may be.

INSERT into *Student* values(103,'Chris');

UPDATE command

Update command is used to update a row of a table. Following is its general syntax,

UPDATE *table-name* set column-name = value *where* condition;

Lets see an example,

update *Student* set age=18 where s_id=102;

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	chris	14

Example to Update multiple columns

UPDATE *Student* set s_name='Abhi',age=17 where s_id=103;

The above command will update two columns of a record.

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

3) Delete command

Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. Following is its general syntax, **DELETE from *table-name*;**

Example to Delete all Records from a Table

DELETE from *Student*;

The above command will delete all the records from **Student** table.

Example to Delete a particular Record from a Table

Consider the following **Student** table

S_id	S_Name	age
101	Adam	15
102	Alex	18
103	Abhi	17

DELETE from *Student* where s_id=103;

The above command will delete the record where s_id is 103 from **Student** table.

S_id	S_Name	age
101	Adam	15
102	Alex	18

TCL command commands are used to manage transactions in database.These are used to manage the changes made by DML statements. It also allows statements to be grouped together into logical transactions.

Commit command

Commit command is used to permanently save any transaaction into database.

Following is Commit command's syntax,

commit;

Rollback command

This command restores the database to last commited state. It is also use with savepoint command to jump to a savepoint in a transaction. Following is Rollback command's syntax,

rollback to *savepoint-name*;

Savepoint command

savepoint command is used to temporarily save a transaction so that you can rollback to that point whenever necessary.

Following is savepoint command's syntax,

savepoint *savepoint-name*;

Example of Savepoint and Rollback

Following is the **class** table,

ID	NAME
1	abhi
2	adam
4	alex

Lets use some SQL queries on the above table and see the results.

INSERT into *class* values(5,'Rahul');

commit;

UPDATE *class* set name='abhijit' where id='5'; savepoint A;

INSERT into *class* values(6,'Chris');

savepoint B;

INSERT into *class* values(7,'Bravo');

savepoint C;

SELECT * from *class*;

The resultant table will look like,

Now **rollback** to **savepoint B**

rollback to B;

SELECT * from *class*;

The resultant table will look like

Now **rollback** to **savepoint A**

rollback to A;

SELECT * from *class*;

The result table will look like

DCL command

Data Control Language(DCL) is used to control privilege in Database. To perform any operation in the database, such as for creating tables, sequences or views we need privileges. Privileges are of two types,

System : creating session, table etc are all types of system privilege.

Object : any command or query to work on tables comes under object privilege. DCL defines two commands,

Grant : Gives user access privileges to database.

Revoke : Take back permissions from user.

To Allow a User to create Session

grant create session to *username*;

To Allow a User to create Table

grant create table to *username*;

To provide User with some Space on Tablespace to store Table

alter user *username* quota unlimited on system;

To Grant all privilege to a User

grant sysdba to *username*

To Grant permission to Create any Table

grant *create* any table to *username*

To Grant permission to Drop any Table

grant *drop* any table to *username*

To take back Permissions

revoke create table from *username*

Data Base Access from Application Programs:

SQL: Application programs generally access databases through one of

–Language extensions to allow embedded SQL

–Application program interface (e.g., ODBC/JDBC) which allow SQL queries to

be sent to a database.

Customer:

Example: Find the name of the customer with customer-id 192-83-7465
SQL>select *customer.customer_name*

Example: Find the balances of all accounts held by the customer with customer-Id 192-83-7465.

SQL>select *account.balance*
from *depositor,account*
where *depositor.customer_id*='192-83-7465'and
depositor.account_number = account.account_number;

Database Architecture:

The architecture of a database systems is greatly influenced by the underlying computer system on which the database is running:

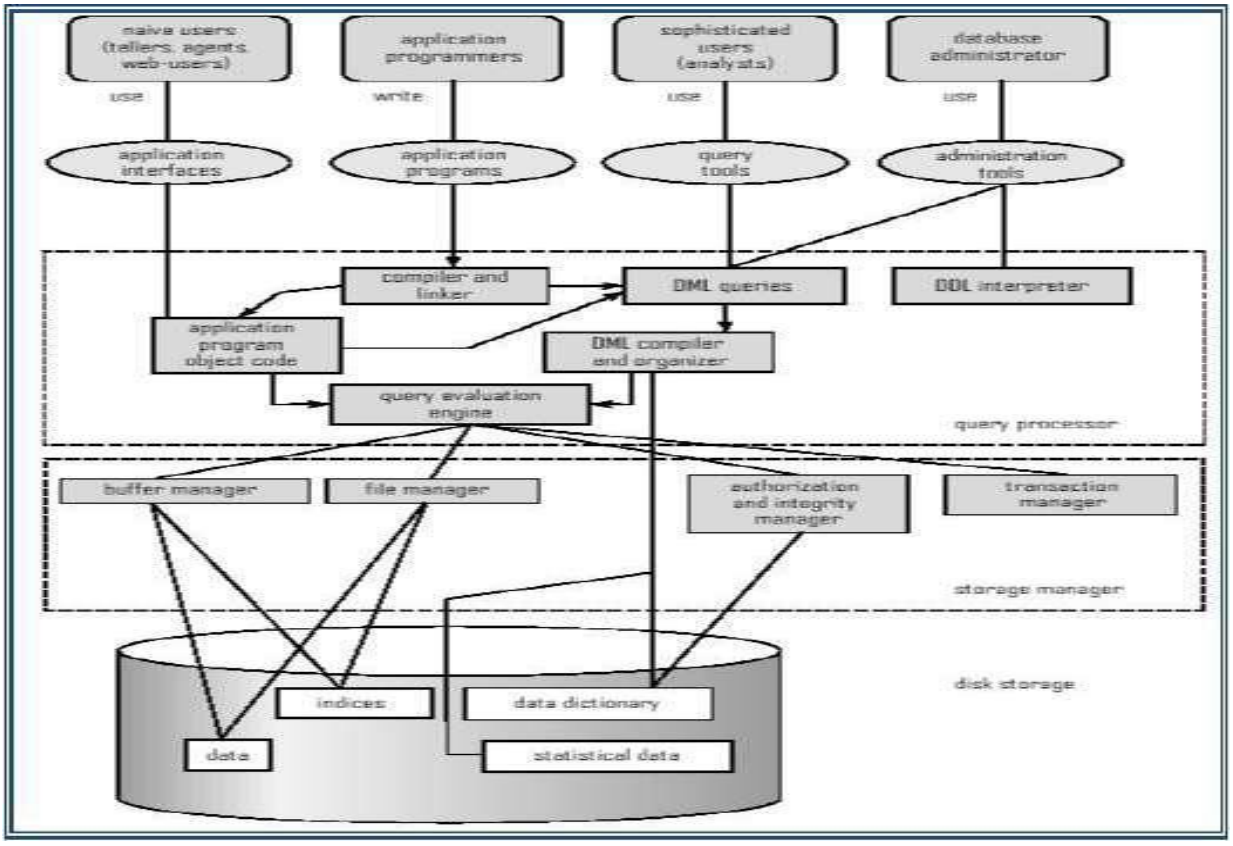
Centralized

Client-server

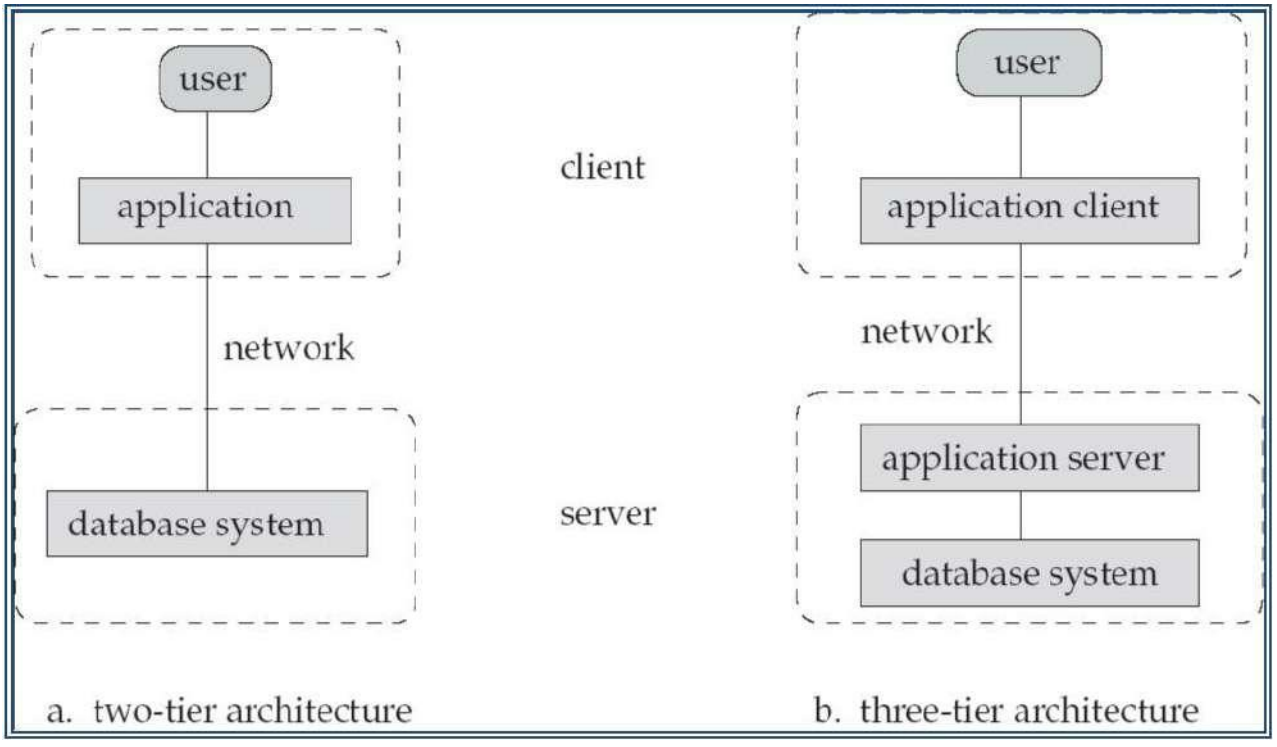
Parallel (multiple processors and disks)

Distributed

Overall System Structure



Database Application Architectures:



Transaction Management:

A **transaction** is a collection of operations that performs a single logical function in a database application A transaction in a database system must maintain atomicity, consistency, isolation, and durability − commonly known as ACID properties properties − in order to ensure accuracy, completeness, and data integrity.

Transaction-management component ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.

Concurrency-control manager controls the interaction among the concurrent transactions, to ensure the consistency of the database.

Data storage and Querying:

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of the database system are

- Storage management
- Query processing
- Transaction processing

Storage Management

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 to the following tasks:

- Interaction with the file manager
- Efficient storing, retrieving and updating of data
- Authorization and integrity manager
- Integrity
- Transaction manager
- File manager
- Buffer manager

Issues:

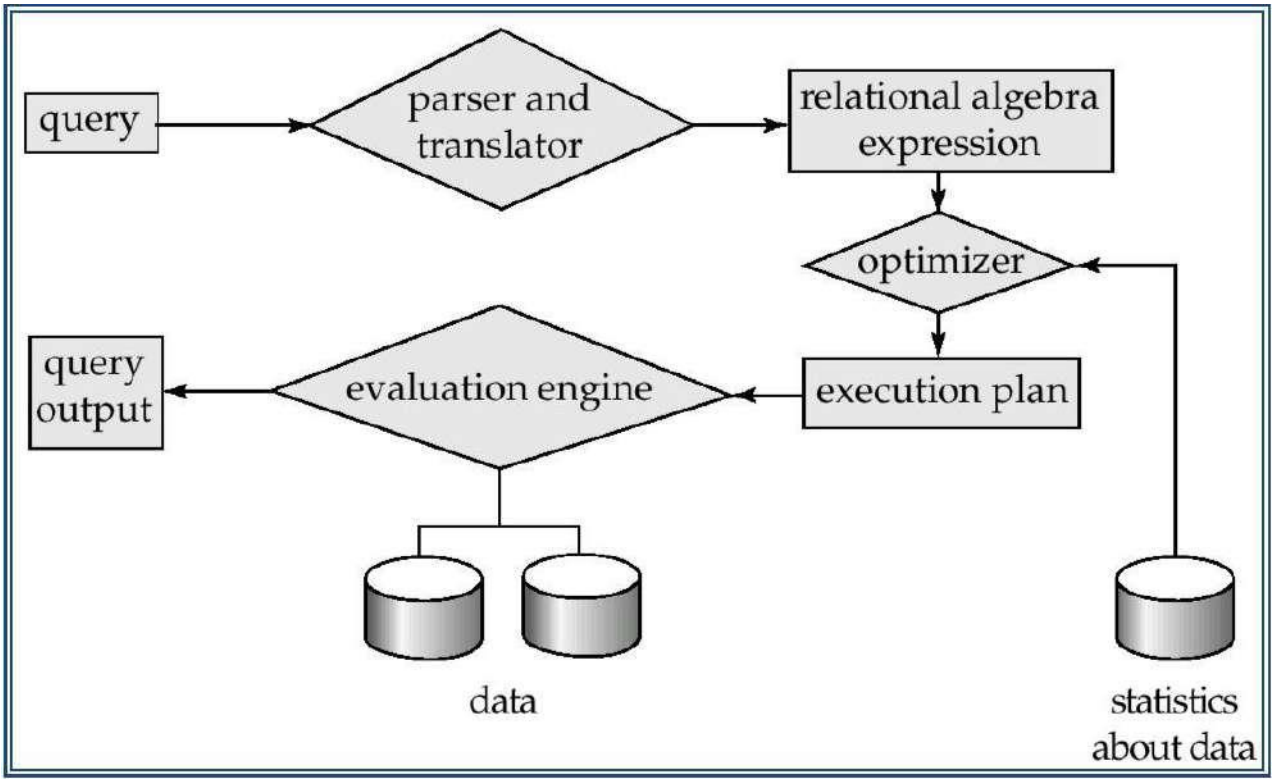
- Storage access
- File organization
- Indexing and hashing

Query Processing

Parsing and translation

Optimization

Evaluation



Alternative ways of evaluating a given query

- Equivalent expressions
- Different algorithms for each operation

Cost difference between a good and a bad way of evaluating a query can be enormous

Need to estimate the cost of operations

– Depends critically on statistical information about relations which the database must maintain

– Need to estimate statistics for intermediate results to compute cost of complex expressions

Database Users and Administrators:

Database Users

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

Application programmers – interact with system through DML calls

Sophisticated users – form requests in a database query language

Specialized users – write specialized database applications that do not fit into the traditional data processing framework

Naïve users – invoke one of the permanent application programs that have been written previously

– Examples, people accessing database over the web, bank tellers, clerical staff

Database Administrator

Coordinates all the activities of the database system

–has a good understanding of the enterprise's information resources and needs.

Database administrator's duties include:

–Storage structure and access method definition

–Schema and physical organization modification

–Granting users authority to access the database

–Backing up data

–Monitoring performance and responding to changes

–Database tuning.