

## Data Base Management Systems

### Data Base Management Systems

#### Chapter 01

##### Overview of Data Base Management Systems:

Data bases and database systems have become an essential component of everyday life in modern society.

Examples for database Applications:

- Purchases from the supermarket
- Purchases using your credit card
- Booking a holiday at the travel agents
- Using the local library
- Taking out insurance
- Using the Internet
- Studying at university

Need to store data :

Data originates at one time and used later(i.e.) Store registrations for grading later, Store for future information needs, Governmental regulations requires access to past data, Data used later for auditing, evaluation purpose, Used more than once : save for future use

Limitations of manual methods:

Problems of speed, Problems of accuracy, Problems of consistency and reliability, Problems of poor response time, Problems of work-load handling capability, Problems of meeting ad hoc information needs, Problems of cost, Problems due to human frailties: (misplaced) loyalty, inconsistency, irregularity, difficulties in handling big tasks ...

Why computerized data processing?

Advantage of speed, Advantage of accuracy, Advantage of reliability and consistency, Advantage of storage and retrieval efficiency, Advantage of on-line-access to meet ad-hoc needs, Advantage of cost

**Data Base** : Collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning.

STUDENT	Name	StudentNumber	Class	Major
	Smith	17	1	CS
	Brown	8	2	CS

COURSE	CourseName	CourseNumber	CreditHours	Department
	Intro to Computer Science	CS1310	4	CS
	Data Structures	CS3320	4	CS
	Discrete Mathematics	MATH2410	3	MATH
	Database	CS3380	3	CS

## Data Base Management Systems

**Definition of DBMS:** A data base management system (DBMS) is a collection of programs that enables users to create and maintain a database. The DBMS is hence a general purpose software system that facilitate the process of defining, constructing, manipulating and sharing databases among the various users and applications.

Historical development of database Technologies:

- **Early Database Applications:** The Hierarchical and Network Models were introduced in mid 1960's and dominated during the seventies. A bulk of the worldwide database processing still occurs using these models.
- **Relational Model based Systems:** The model that was originally introduced in 1970 was heavily researched and experimented with in IBM and the universities. Relational DBMS Products emerged in the 1980's.
- **Object-oriented applications:** OODBMSs were introduced in late 1980's and early 1990's to cater to the need of complex data processing in CAD and other applications. Their use has not taken off much.
- **Data on the Web and E-commerce Applications:** Web contains data in HTML (Hypertext markup language) with links among pages. This has given rise to a new set of applications and E-commerce is using new standards like XML (eXtended Markup Language).

### Extending Database Capabilities:

New functionality is being added to DBMSs in the following areas:

Scientific Applications, Image Storage and Management, Audio and Video data management, Data Mining, Spatial data management, Time Series and Historical Data Management

### Transaction Management:

- A transaction is a collection of operations that performs a single logical function in a database application
- 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.

A database transaction is a unit of interaction with a database management system or similar system that is treated in a coherent and reliable way independent of other transactions that must be either entirely completed or aborted.

In some systems, transactions are also called **LUW** for Logical Units of Work.

In database products the ability to handle transactions allows

## Data Base Management Systems

A single transaction might require several queries, each reading and/or writing information in the database. When this happens it is usually important to be sure that the database is not left with only some of the queries carried out. For example, when doing a money transfer, if the money was debited from one account, it is important that it also be credited to the depositing account. Also, transactions should not interfere with each other.

### **Storage Management:**

- Storage management 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

### **Database Administrator:**

A database administrator (DBA) is a person who is responsible for the environmental aspects of a database. In general, these include:

- Recoverability - Creating and testing Backups
- Integrity - Verifying or helping to verify data integrity
- Security - Defining and/or implementing access controls to the data
- Availability - Ensuring maximum uptime
- Performance - Ensuring maximum performance given budgetary constraints
- Development and testing support - Helping programmers and engineers to efficiently utilize the database.

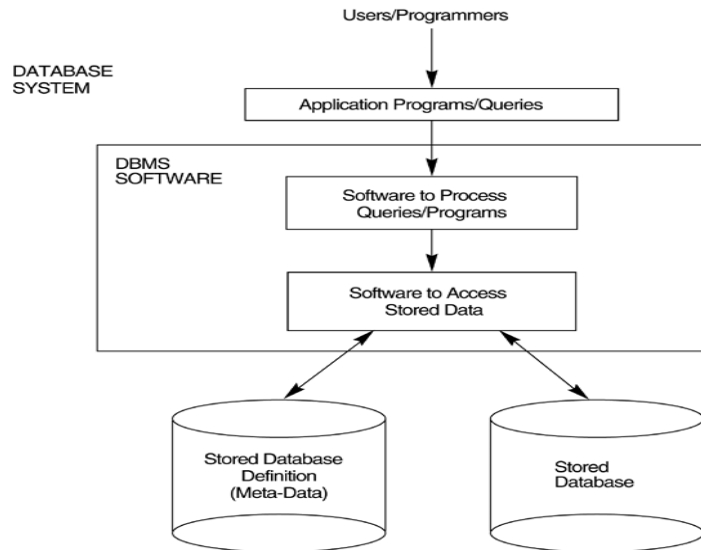
The role of a database administrator has changed according to the technology of database management systems (DBMSs) as well as the needs of the owners of the databases.

### **Types of Databases and Database Applications**

- Numeric and Textual Databases (Traditional Database)
- Multimedia Databases (Video clips, pictures, sound message)
- Geographic Information Systems (GIS) (Weather data, map analysis, satellite images)
- Data Warehouses (Decision making)
- Real-time and Active Databases (Internet based (World wide web))

A simplified database system environment.

## Data Base Management Systems



### Advantages of Using the Database Approach

- Controlling redundancy in data storage and in development and maintenance efforts.
  - Duplication is wasteful. It costs time and money to enter the data more than once.
  - It takes up additional storage space, again with associated costs.
  - Perhaps more importantly, duplication can lead to loss of data integrity.
- Sharing of data among multiple users.
- Restricting unauthorized access to data.
  - When multiple users shares a large database, it is likely that most users will not be authorized to access all information in the database.
- Providing persistent storage for program Objects
  - A complex object in C++ can be stored permanently in an Object Oriented DBMS. Such an object is said to be persistence, since it survives the termination of the program execution and can later be directly retrieved by another C++ program.
- Providing backup and recovery services.
- Providing multiple interfaces to different classes of users.
- Representing complex relationships among data.

### Database Users:

- **Database administrators:** responsible for authorizing access to the database, for co-ordinating and monitoring its use, acquiring software, and hardware resources, controlling its use and monitoring efficiency of operations.
- **Database Designers:** Responsible to define the content, the structure, the constraints, and functions or

## Data Base Management Systems

transactions against the database. They must communicate with the end-users and understand their needs.

- **End users** : End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use. There are several categories of end users:
  - **Casual End User**: access database occasionally when needed. But they may need different information each time.
  - **Naïve or Parametric End user** : they make up a large section of the end-user population. They use previously well-defined functions in the form of "canned transactions" against the database. Examples are bank-tellers or reservation clerks who do this activity for an entire shift of operations.
  - **Sophisticated End User** : These include business analysts, scientists, engineers, others thoroughly familiar with the system capabilities. Many use tools in the form of software packages that work closely with the stored database.
  - **Stand-alone End User** : Mostly maintain personal databases using ready-to-use packaged applications. An example is a tax program user that creates his or her own internal database.

### Describing data : Levels of Abstraction

- **Database Schema**: The *description* of a database. Includes descriptions of the database structure and the constraints that should hold on the database.
- **Schema Diagram**: A diagrammatic display of (some aspects of) a database schema.

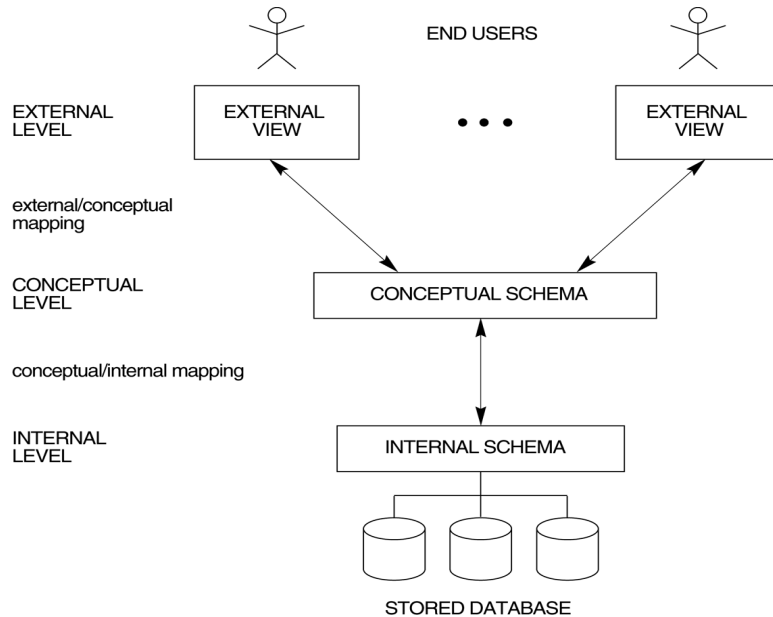
The data in a DBMS is described at three levels of abstraction, as illustrated in the figure.

Defines DBMS schemas at *three levels*:

- **Internal schema** at the internal level to describe physical storage structures and access paths. Typically uses a *physical* data model. (how a record (e.g., customer) is stored-Physical level)
- **Conceptual schema** at the conceptual level to describe the structure and constraints for the *whole* database for a community of users. Uses a *conceptual* or an *implementation* data model. (describes data stored in database, and the relationships among the data - Logical level)
- **External schemas** at the external level to describe the various user views. *www.jntuworld.com* uses the same data model as

## Data Base Management Systems

the conceptual level. (describes data as seen by a user/application - View Level)



- **Schema** - describes contents of the database
  - e.g., what information about a set of customers and accounts and the relationship between them)
  - **Physical schema:** how data is stored at physical level (how)
  - **Logical schema:** data contained at the logical level (what)
- **Database Instance:** The actual data stored in a database at a *particular moment in time*. Also called database state (or occurrence).

### Data Independence :

When a schema at a lower level is changed, only the **mappings** between this schema and higher-level schemas need to be changed in a DBMS that fully supports data independence. The higher-level schemas themselves are *unchanged*. Hence, the application programs need not be changed since they refer to the external schemas

- **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.
- **Logical Data Independence** - the ability to modify conceptual schema without changing the external Schema or application programs.

## Data Base Management Systems

### History of Data Models :

- Relational Model: proposed in 1970 by E.F. Codd (IBM), first commercial system in 1981-82. Now in several commercial products (DB2, ORACLE, SQL Server, SYBASE, INFORMIX).
- Network Model: the first one to be implemented by Honeywell in 1964-65 (IDS System). Adopted heavily due to the support by CODASYL (CODASYL - DBTG report of 1971). Later implemented in a large variety of systems - IDMS (Cullinet - now CA), DMS 1100 (Unisys), IMAGE (H.P.), VAX -DBMS (Digital Equipment Corp.).
- Hierarchical Data Model: implemented in a joint effort by IBM and North American Rockwell around 1965. Resulted in the IMS family of systems. The most popular model. Other system based on this model: System 2k (SAS inc.)
- Object-oriented Data Model(s): several models have been proposed for implementing in a database system. One set comprises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE). Additionally, systems like O2, ORION (at MCC - then ITASCA), IRIS (at H.P.- used in Open OODB).
- Object-Relational Models: Most Recent Trend. Started with Informix Universal Server. Exemplified in the latest versions of Oracle-10i, DB2, and SQL Server etc. systems.

## Data Base Management Systems

### Chapter 02

#### Entity Relation Model:

The Entity Relationship (ER) data model allows us to describe the data involved in a real world enterprise in terms of objects and their relationships and is widely used to develop an initial data base design. Within the larger context of the overall design process, the ER model is used in a phase called "*Conceptual database design*".

#### Database design and ER Diagrams:

The database design process can be divided into six steps. The ER model is most relevant to the first three steps.

##### 1. Requirement Analysis:

The very first step in designing a database application is to understand what data is to be stored in the database, what application must be built in top of it, and what operations are most frequent and subject to performance requirements. In other words, we must find out what the users want from the database.

##### 2. Conceptual database Design:

The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints known to hold over this data. The ER model is one of several high-level or semantic, data models used in database design.

##### 3. Logical Database Design:

We must choose a database to convert the conceptual database design into a database schema in the data model of the chosen DBMS. Normally we will consider the Relational DBMS and therefore, the task in the logical design step is to convert an ER schema into a relational database schema.

#### **Beyond ER Design**

##### 4. Schema Refinement:

This step is to analyze the collection of relations in our relational database schema to identify potential problems, and refine it.

##### 5. Physical Database Design:

This step may simply involve building indexes on some table and clustering some tables or it may involve substantial redesign of parts of database schema obtained from the earlier steps.



## Data Base Management Systems

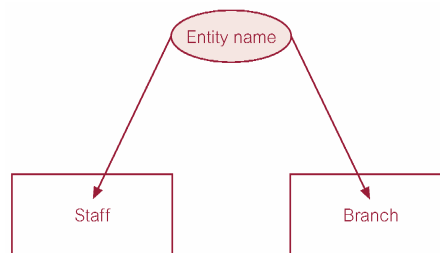
### 6. Application and security Design:

Any software project that involves a DBMS must consider aspects of the application that go beyond the database itself. We must describe the role of each entity (users, user group, departments) in every process that is reflected in some application task, as part of a complete workflow for the task. A DBMS Provides several mechanisms to assist in this step.

### Entity Types, Attributes and Keys:

**Entities are specific objects** or things in the mini-world that are represented in the database.

For example, Employee or staff, Department or Branch , Project are called Entity.



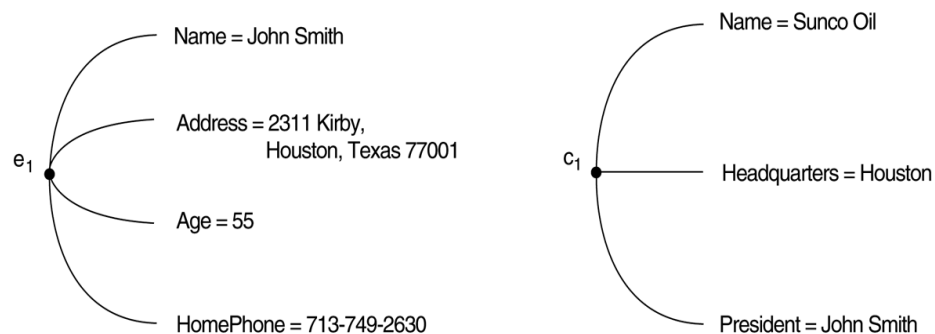
### **Attributes are properties used to describe an entity.**

For example an EMPLOYEE entity may have a Name, SSN, Address, Sex, BirthDate and Department may have a Dname, Dno, DLocation.

A specific entity will have a value for each of its attributes.

For example a specific employee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55'

Each attribute has a *value set* (or data type) associated with it - e.g. integer, string, subrange, enumerated type, ...



## Data Base Management Systems

### Types of Attributes:

- Simple (Atomic) Vs. Composite
- Single Valued Vs. Multi Valued
- Stored Vs. Derived
- Null Values

### Simple Vs. Composite Attribute:

- Simple
  - Attribute that are not divisible are called simple or atomic attribute.
  - For example, Street\_name or Door\_number. i.e. the division of composite attribute.
- Composite
  - The attribute may be composed of several components.
  - For example, Address (Apt#, House#, Street, City, State, ZipCode, Country) or Name (FirstName, MiddleName, LastName). Composition may form a hierarchy where some components are themselves composite.

### Single Valued Vs. Multi Valued:

- Single Valued
  - An attribute having only one value.
  - For example, Age, Date of birth, Sex, SSN
- Multi-valued
  - An entity may have multiple values for that attribute.
  - For example, Color of a CAR or Previous Degrees of a STUDENT. Denoted as {Color} or {PreviousDegrees}. Phone number of an employee.

### Stored Vs. Derived

In some cases two or more attributes values are related - for example the age and date of birth of a person. For a particular person entity, the value of age can be determined from the current (today's) date and the value of that person's Birthdate.

The Age attribute is hence called a **derived attribute** and is said to be derivable from the Birthdate attribute, which is called stored attribute.

### Null Values

In some cases a particular entity may not have an applicable value for an attribute. For example, a college degree attribute applies only to persons with college degrees. For such situation, a special value called **null** is created.

### Key attributes of an Entity Type:

An important constraint on the entities of an entity type is the **Key or uniqueness constraint** on attributes. An entity type usually has an attribute whose values are distinct for

## Data Base Management Systems

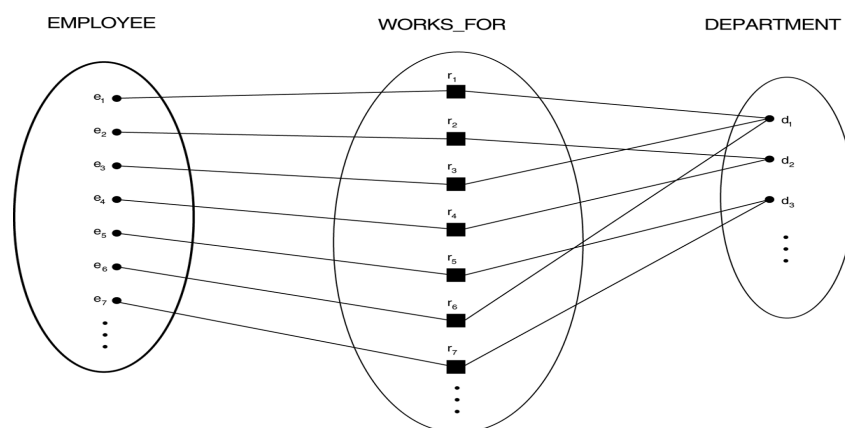
each individual entity in the entity set. Such an attribute is called a **Key attribute**, and its values can be used to identify each entity uniquely.

For example name attribute is a key of the company entity type, because no two companies are allowed to have the same name. For the person entity type, a typical key attribute is socialSecurityNumber (SSN). In ER diagrammatic notation, each key attribute has its name underlined inside the oval.

### Relationships and Relationship sets:

A relationship is an association among two or more entities. For example we may have the relationship that Antony works in the Marketing department. A **relationship type**  $R$  among the  $n$  entity types  $E_1, E_2, \dots, E_n$  defines a set of associations or a **relationship set**— among entities from these entity types.

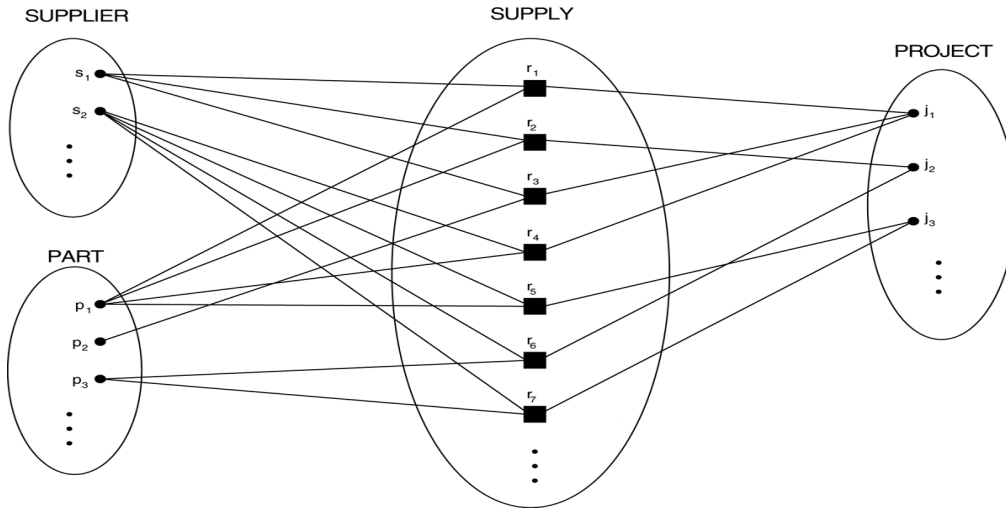
Informally each relationship instance  $r_i$  in  $R$  is an association of entities, where the association includes exactly one entity from each participating entity type. Each such relationship instance  $r_i$  represents the facts that the entities participating in  $r_i$  are related in some way in the corresponding mini world situation. For example consider a relationship type *Works\_for* between the two entity types Employee and Department, which associates each employee with the department for which the employee works. Each relationship instance in the relationship set *Works\_for* associates one employee entity and one department entity. Figure illustrates this example.



### Degree of a Relationship:

The degree of a relationship is the number of participating entity types. Hence the above work\_for relationship is of degree two. A relationship type of degree two is called **Binary**, and one of degree three is called **Ternary**. An example of Ternary relationship is given below.

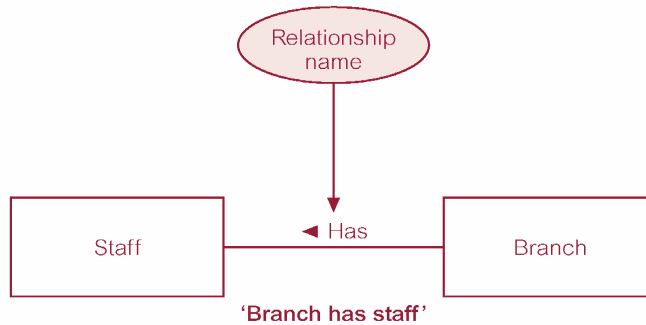
### Data Base Management Systems



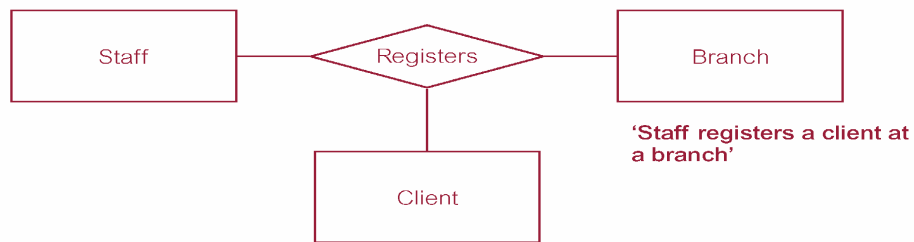
Relationship of degree:

- two is binary
- three is ternary
- four is quaternary

Another Example for Binary Relationship

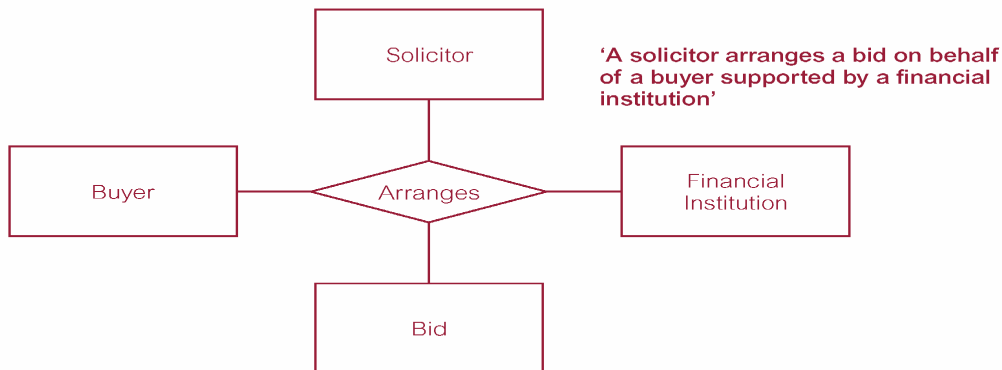


Example for Ternary Relationship.



## Data Base Management Systems

Example for quaternary Relationship



Constraints on relationship Types:

Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in the corresponding relationship set. These constraints are determined from the mini world situation that the relationship represent. For example in the works\_for relationship, if the company has a rule that each employee must work for exactly one department, that we would like to describe this constraint in the schema. There are two type.

### 1. Cardinality Ratio

Describes maximum number of possible relationship occurrences for an entity participating in a given relationship type.

### 2. Participation:

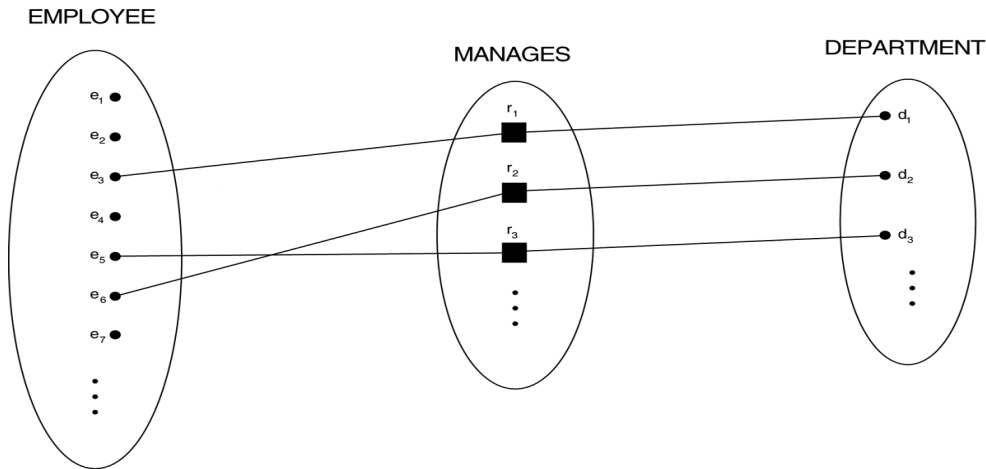
Determines whether all or only some entity occurrences participate in a relationship.

- **Cardinality ratio** (of a binary relationship): 1:1, 1:N, N:1, or M:N  
**SHOWN BY PLACING APPROPRIATE NUMBER ON THE LINK.**
- **Participation constraint** (on each participating entity type): total (called *existence dependency*) or partial.  
**SHOWN BY DOUBLE LINING THE LINK**

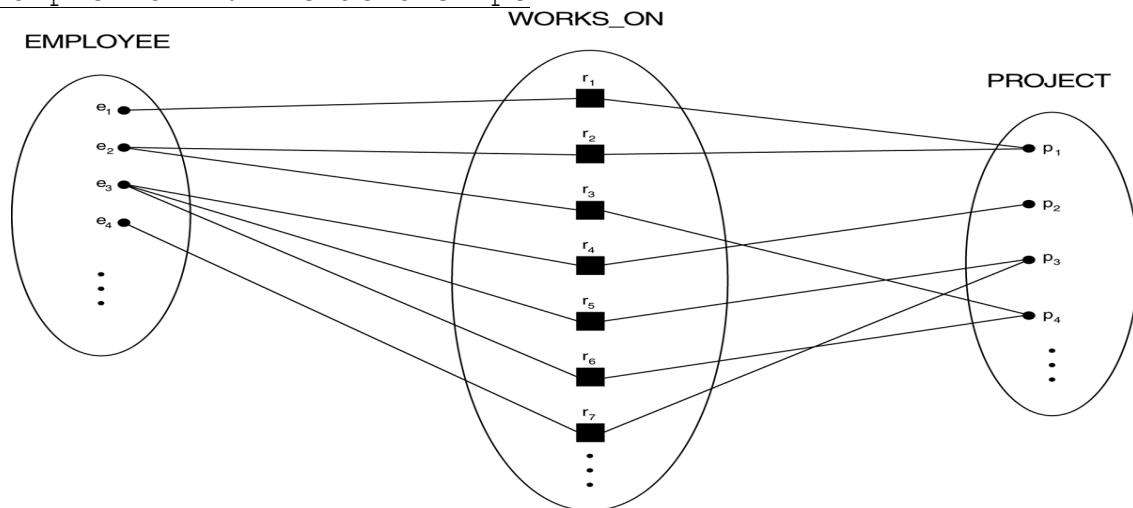
NOTE: These are easy to specify for Binary Relationship Types.

## Data Base Management Systems

### Example for 1:1 relationship



### Example for M:N relationships



### Alternative (min, max) notation for relationship structural constraints:

- Specified on *each participation* of an entity type E in a relationship type R
- Specifies that each entity e in E participates in *at least* min and *at most* max relationship instances in R
- Default(no constraint): min=0, max=n
- Must have min ≤ max, min ≥ 0, max ≥ 1
- Derived from the knowledge of mini-world constraints

Examples:

- A department has *exactly one* manager and an employee can manage *at most one* department.

– Specify (0,1) for participation of EMPLOYEE in MANAGES

– Specify (1,1) for participation of DEPARTMENT in MANAGES

## Data Base Management Systems

- An employee can work for *exactly one* department but a department can have *any number of employees*.
  - Specify (1,1) for participation of EMPLOYEE in WORKS\_FOR
  - Specify (0,n) for participation of DEPARTMENT in WORKS\_FOR

### Types of Entity:

1. Strong Entity
  - a. Entity which has a key attribute in its attribute list.
2. Weak Entity
  - a. Entity which doesn't have the Key attribute.

### **Weak Entity Sets:**

An entity set that does not possess sufficient attributes to form a primary key is called a weak entity set. One that does have a primary key is called a strong entity set.

For example,

- The entity set transaction has attributes transaction-number, date and amount.
- Different transactions on different accounts could share the same number.
- These are not sufficient to form a primary key (uniquely identify a transaction).
- Thus transaction is a weak entity set.

For a weak entity set to be meaningful, it must be part of a one-to-many relationship set. This relationship set should have no descriptive attributes.

- Member of a strong entity set is a dominant entity.
- Member of a weak entity set is a subordinate entity.

A weak entity set does not have a primary key, but we need a means of distinguishing among the entities.

The discriminator of a weak entity set is a set of attributes that allows this distinction to be made.

The primary key of a weak entity set is formed by taking the primary key of the strong entity set on which its existence depends (see Mapping Constraints) plus its discriminator.

To illustrate:

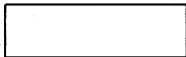
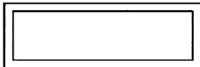

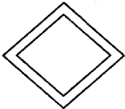
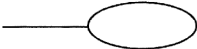


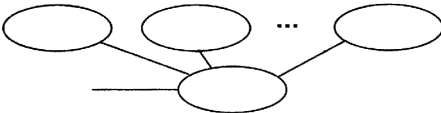
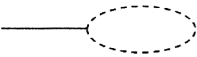
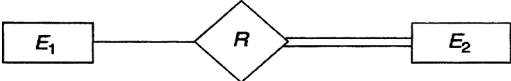
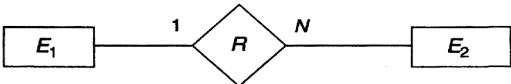
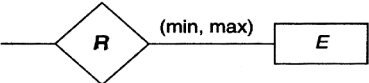
- Transaction is a weak entity. It is existence-dependent on account.
- The primary key of account is account-number.
- Transaction-number distinguishes transaction entities within the same account (and is thus the discriminator).
- So the primary key for transaction would be (account-number, transaction-number).

**Note:** The primary key of a weak entity is found by taking the primary key of the strong entity on which it is existence-dependent, plus the discriminator of the weak entity set.

## Data Base Management Systems

### ER Diagram

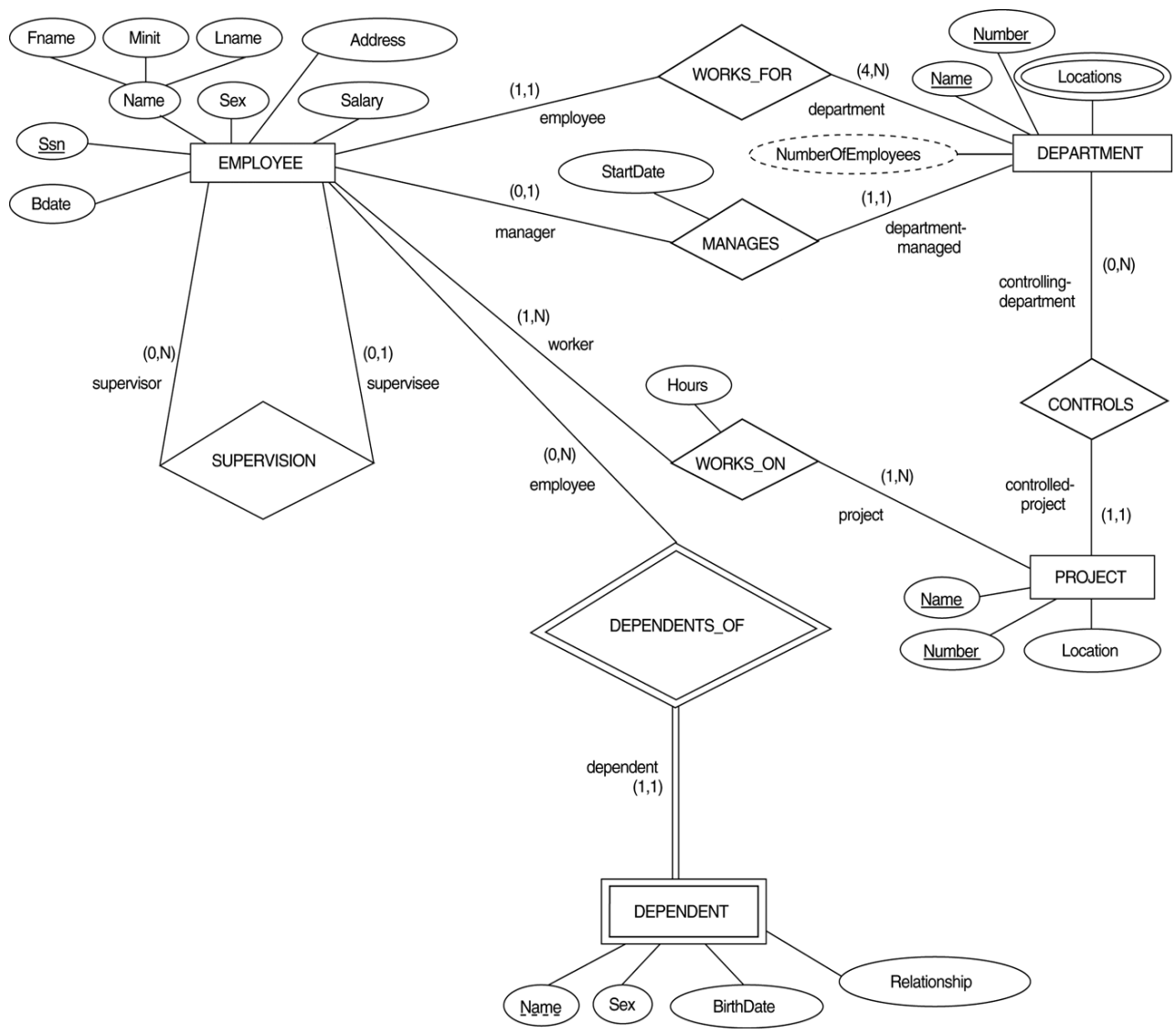
Notations for ER Diagram

<u>Symbol</u>	<u>Meaning</u>
	ENTITY
	WEAK ENTITY
	RELATIONSHIP
	IDENTIFYING RELATIONSHIP
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF $E_2$ IN $R$
	CARDINALITY RATIO 1: $N$ FOR $E_1:E_2$ IN $R$
	STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF $E$ IN $R$

Sample ER diagram for a Company Schema with structural Constraints is shown below.



### Data Base Management Systems

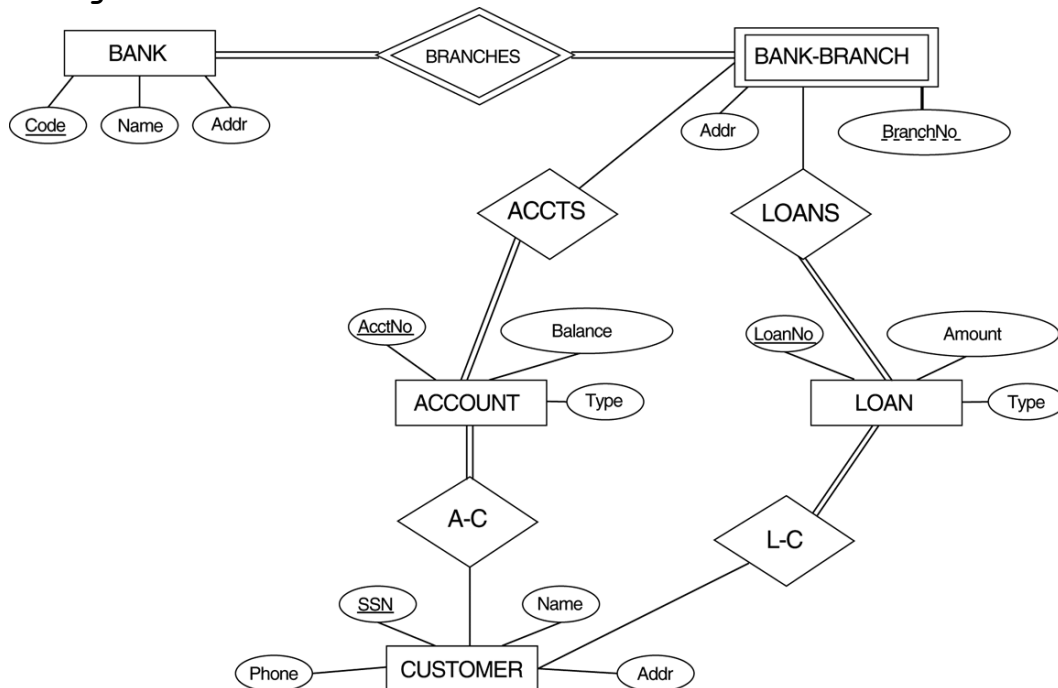


#### Identifying Relationship:

It is a relationship between Strong entity and weak entity.

## Data Base Management Systems

### An ER Diagram for a Bank database Schema



### Enhanced-ER (EER) Model Concepts

- It Includes all modeling concepts of basic ER
- Additional concepts: subclasses/super classes, specialization/generalization,
- The resulting model is called the enhanced-ER or Extended ER (E2R or EER) model
- It is used to model applications more completely and accurately if needed
- It includes some object-oriented concepts, such as inheritance

### Subclasses and Super classes:

- An entity type may have additional meaningful sub groupings of its entities
- Example: EMPLOYEE may be further grouped into SECRETARY, ENGINEER, MANAGER, TECHNICIAN, SALARIED\_EMPLOYEE, HOURLY\_EMPLOYEE, ...
  - Each of these groupings is a subset of EMPLOYEE entities
  - Each is called a subclass of EMPLOYEE
  - EMPLOYEE is the super class for each of these subclasses
- These are called super class/subclass relationships

## Data Base Management Systems

### Specialization:

Is the process of defining a set of subclasses of a super class. The set of subclasses is based upon some distinguishing characteristics of the entities in the super class

For Example,

**{SECRETARY, ENGINEER, TECHNICIAN}**

is a specialization of EMPLOYEE based upon *job type*.

Another specialization of EMPLOYEE based on the *method of pay* is,

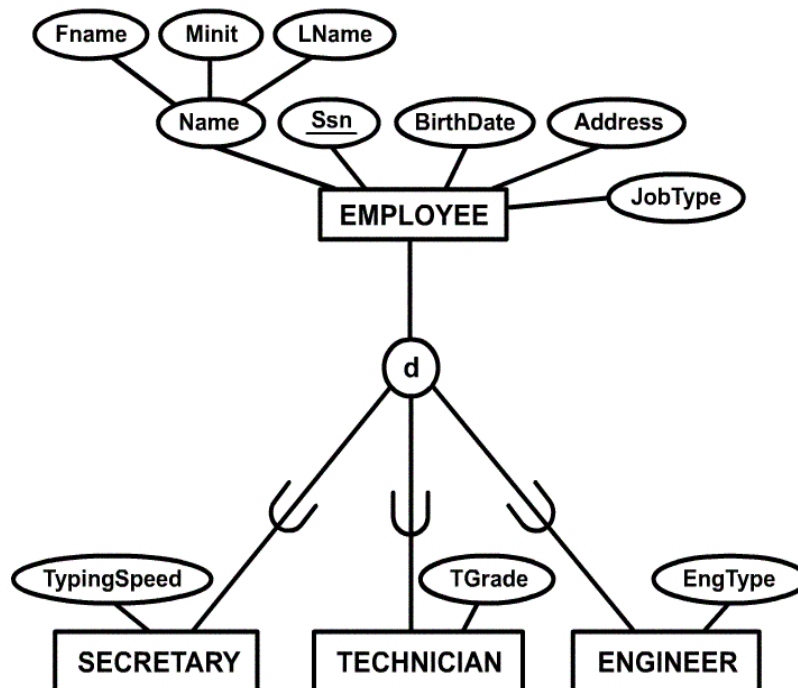
**{SALARIED\_EMPLOYEE, HOURLY\_EMPLOYEE}**.

Super class/subclass relationships and specialization can be diagrammatically represented in EER diagrams. Attributes of a subclass are called **specific attributes**.

For example, Typing Speed of SECRETARY

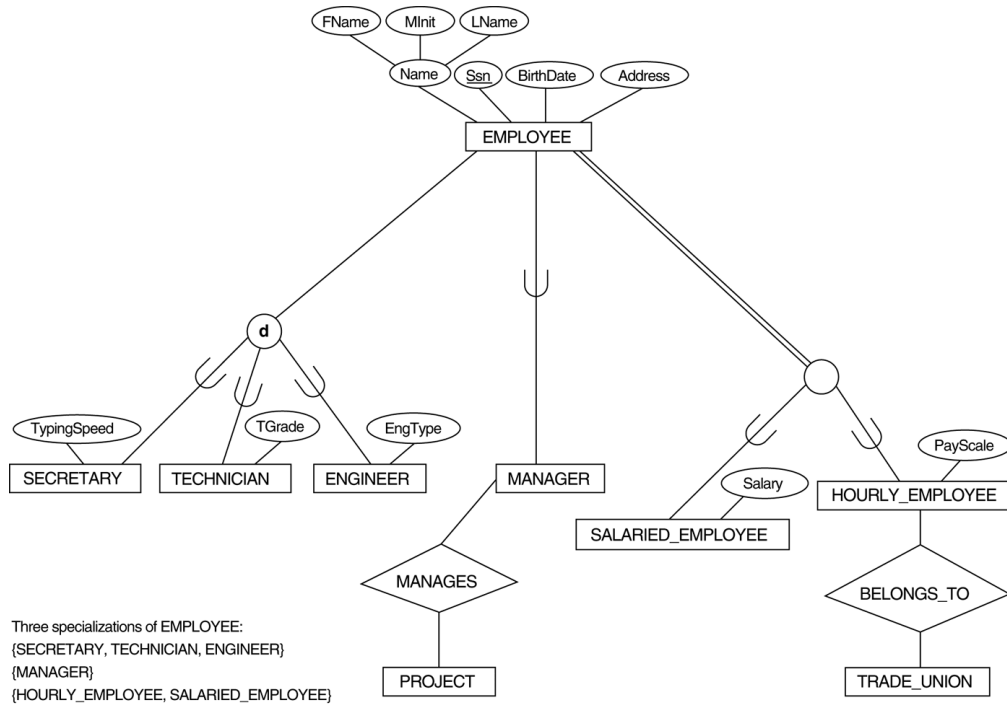
The subclass can participate in specific relationship types. For example, BELONGS\_TO of HOURLY\_EMPLOYEE

**Figure shows the Specialization of an Employee based on Job Type.**



## Data Base Management Systems

We may have several specializations of the same super class



### Generalization:

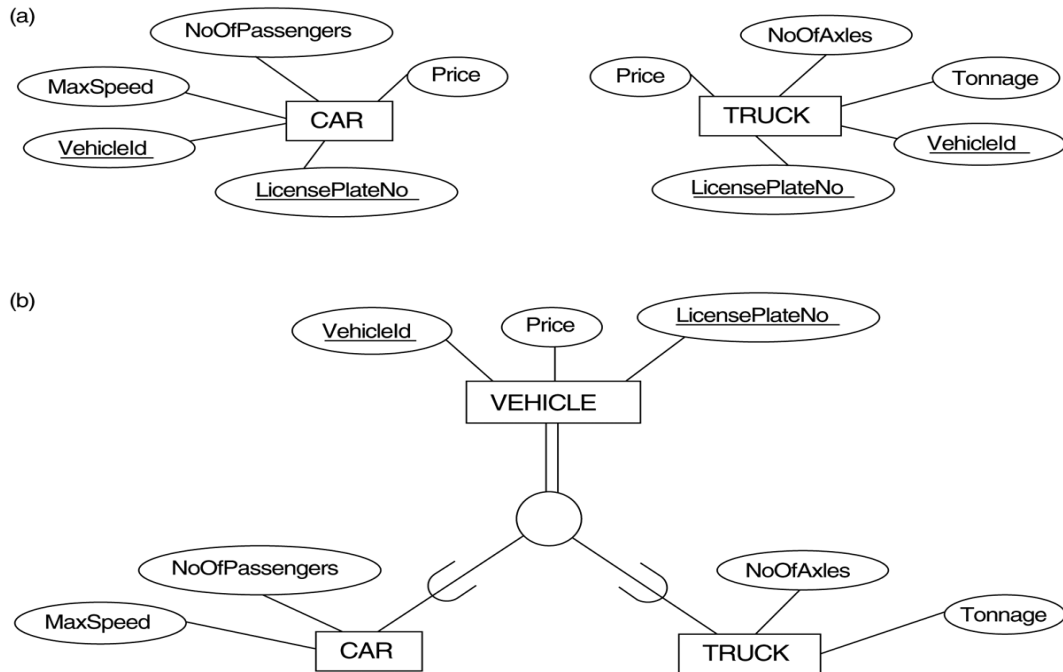
- The reverse of the specialization process
- Several classes with common features are generalized into a super class; original classes become its subclasses
- Example: CAR, TRUCK generalized into VEHICLE; both CAR, TRUCK become subclasses of the super class VEHICLE.
  - We can view {CAR, TRUCK} as a specialization of VEHICLE
  - Alternatively, we can view VEHICLE as a generalization of CAR and TRUCK

**FIGURE** Generalization.

(a) Two entity types, CAR and TRUCK.

(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

## Data Base Management Systems



### Constraints on Specialization and generalization

#### 1. Predicate Defined

- If we can determine exactly those entities that will become members of each subclass by a condition, the subclasses are called **predicate-defined** (or condition-defined) subclasses
  - Condition is a constraint that determines subclass members
  - Display a predicate-defined subclass by writing the predicate condition next to the line attaching the subclass to its super class

#### 2. Attribute Defined:

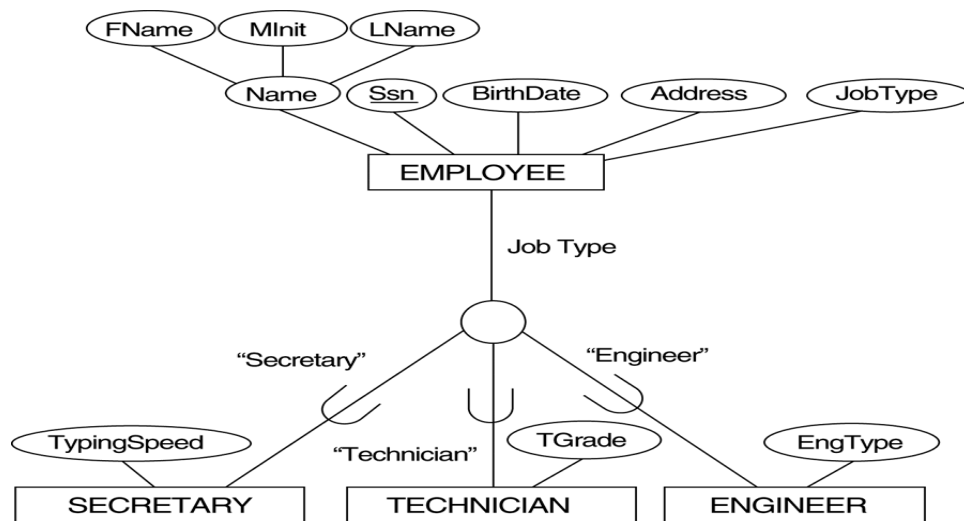
- If all subclasses in a specialization have membership condition on same attribute of the super class, specialization is called an **attribute defined-specialization**
- Attribute is called the defining attribute of the specialization
- Example: Job Type is the defining attribute of the specialization {SECRETARY, TECHNICIAN, ENGINEER} of EMPLOYEE

#### 3. User Defined

## Data Base Management Systems

- subclass is called **user-defined**
  - Membership in a subclass is determined by the database users by applying an operation to add an entity to the subclass
  - Membership in the subclass is specified individually for each entity in the superclass by the user

The figure shows the constraints on Specialization & Generalization



#### 4. Disjointness Constraint:

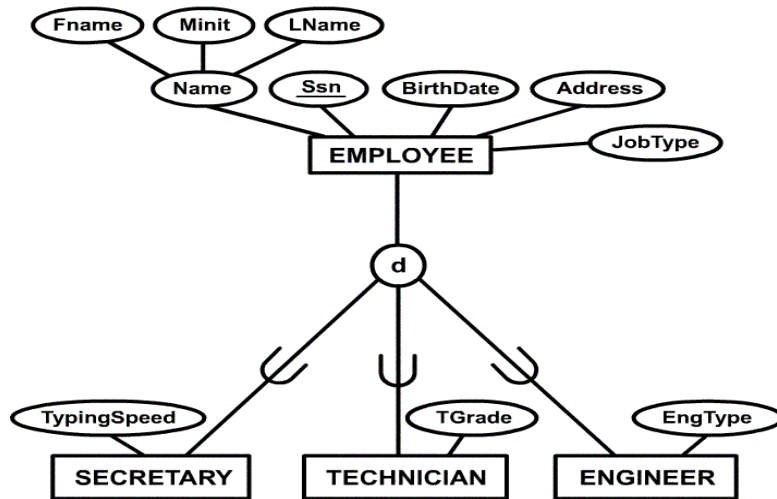
- Specifies that the subclasses of the specialization must be disjoint (an entity can be a member of at most one of the subclasses of the specialization)
- Specified by 'd' in EER diagram
- If not disjoint, overlap; that is the same entity may be a member of more than one subclass of the specialization
- Specified by 'o' in EER diagram

#### 5. Completeness Constraint:

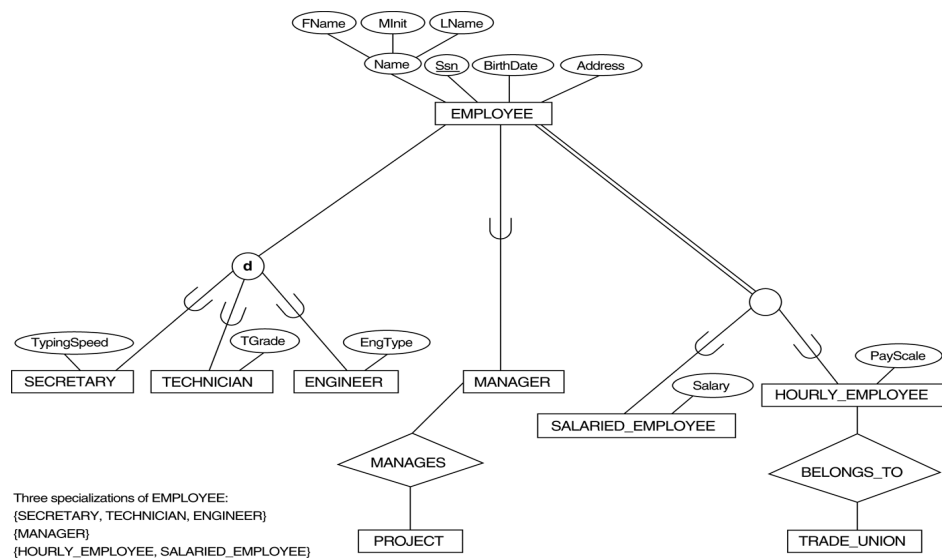
- Total specifies that every entity in the superclass must be a member of some subclass in the specialization/generalization
- Shown in EER diagrams by a **double line**
- Partial allows an entity not to belong to any of the subclasses
- Shown in EER diagrams by a **single line**

## Data Base Management Systems

### Example of disjoint partial Specialization



- Hence, we have four types of specialization / generalization:
  - o Disjoint, total
  - o Disjoint, partial
  - o Overlapping, total
  - o Overlapping, partial
- Note: Generalization usually is total because the super class is derived from the subclasses.**



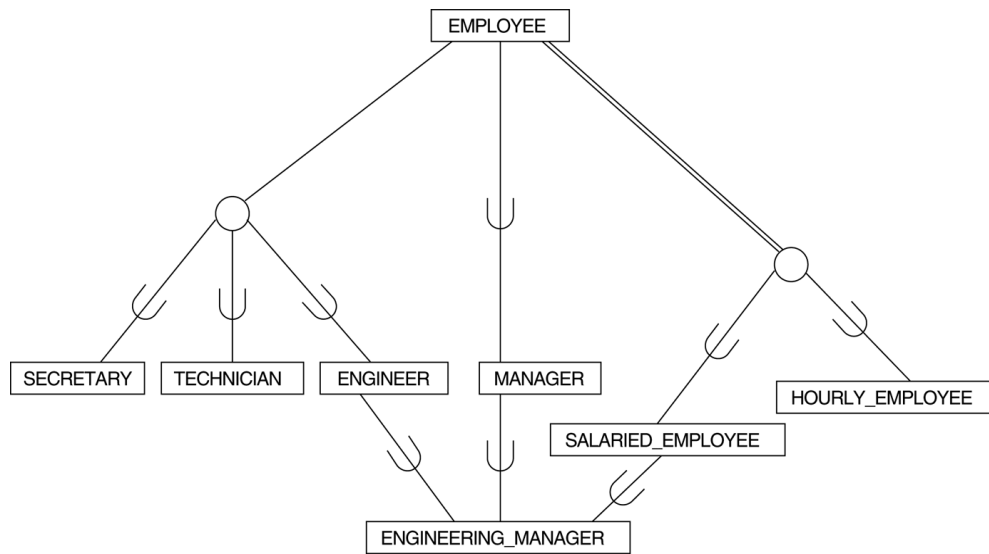
### **Specialization / Generalization Hierarchies, Lattices and Shared Subclasses:**

- A subclass may itself have further subclasses specified on it
- Forms a hierarchy or a lattice

## Data Base Management Systems

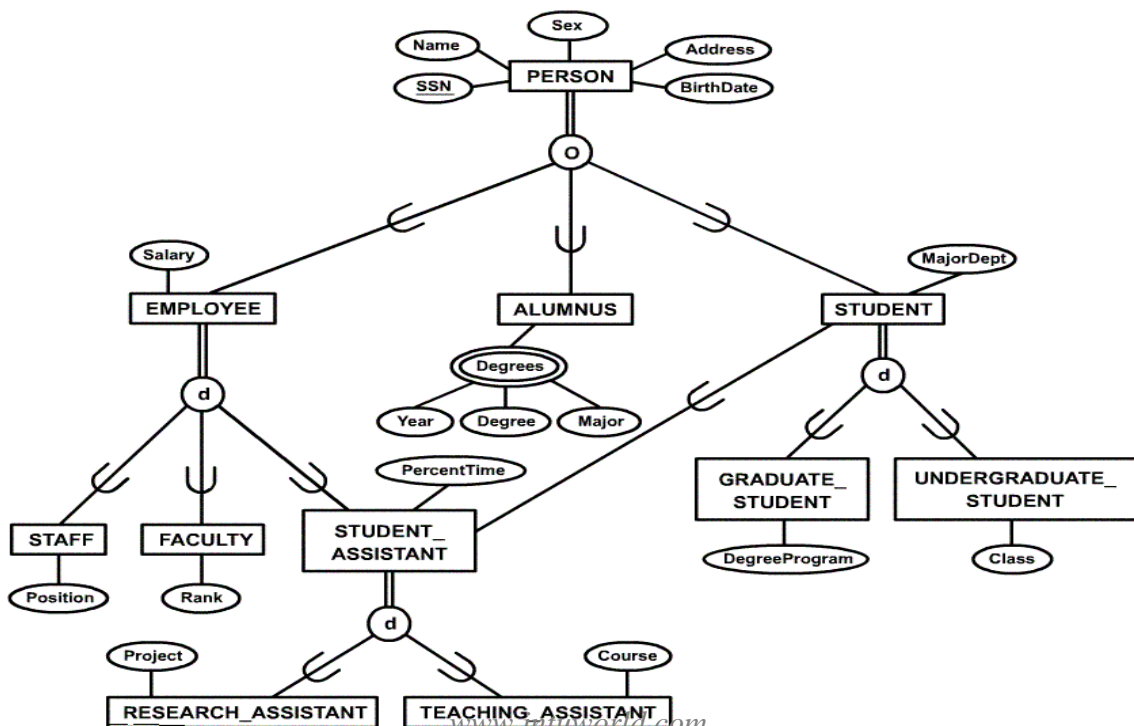
- **Hierarchy** has a constraint that every subclass has only one superclass (called *single inheritance*)
- In a **lattice**, a subclass can be subclass of more than one superclass (called *multiple inheritance*)
- In a lattice or hierarchy, a subclass inherits attributes not only of its direct superclass, but also of all its predecessor superclasses

### A specialization lattice with shared subclass ENGINEERING\_MANAGER



### An EER Diagram Example

### Specialization / Generalization Lattice Example (UNIVERSITY)





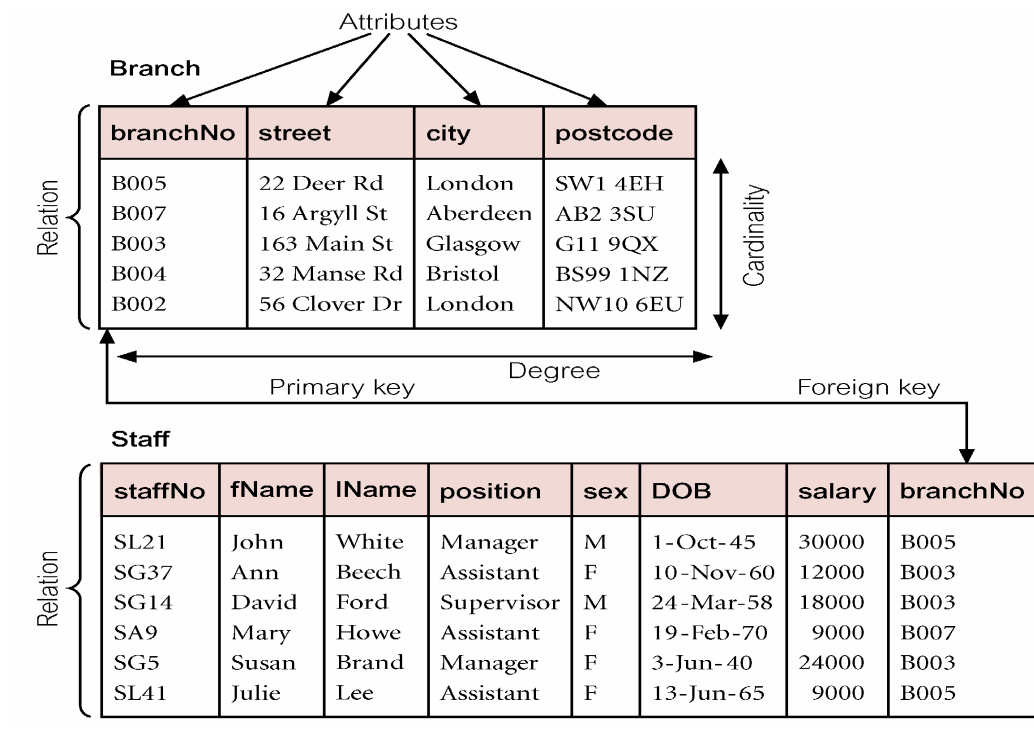
## Data Base Management Systems

### Chapter 03

#### Relational Model

##### Relational Model Terminology :

- ◆ A relation is a table with columns and rows.
  - Only applies to logical structure of the database, not the physical structure.
- ◆ Attribute is a named column of a relation.
- ◆ Domain is the set of allowable values for one or more attributes.
- ◆ Tuple is a row of a relation.
- ◆ Degree is the number of attributes in a relation.
- ◆ Cardinality is the number of tuples in a relation.
- ◆ Relational Database is a collection of normalized relations with distinct relation names.



## Data Base Management Systems

### Examples of Attribute Domains

Attribute	Domain Name	Meaning	Domain Definition
branchNo	BranchNumbers	The set of all possible branch numbers	character: size 4, range B001–B999
street	StreetNames	The set of all street names in Britain	character: size 25
city	CityNames	The set of all city names in Britain	character: size 15
postcode	Postcodes	The set of all postcodes in Britain	character: size 8
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible values of staff birth dates	date, range from 1-Jan-20, format dd-mmm-yy
salary	Salaries	Possible values of staff salaries	monetary: 7 digits, range 6000.00–40000.00

### Alternative Terminology for Relational Model

**Table 3.1** Alternative terminology for relational model terms.

Formal terms	Alternative 1	Alternative 2
Relation	Table	File
Tuple	Row	Record
Attribute	Column	Field

### Mathematical Definition of Relation

- ◆ Consider two sets,  $D_1$  &  $D_2$ , where  $D_1 = \{2, 4\}$  and  $D_2 = \{1, 3, 5\}$ .
- ◆ Cartesian product,  $D_1 \times D_2$ , is set of all ordered pairs, where first element is member of  $D_1$  and second element is member of  $D_2$ .

$$D_1 \times D_2 = \{(2, 1), (2, 3), (2, 5), (4, 1), (4, 3), (4, 5)\}$$

- ◆ Alternative way is to find all combinations of elements with first from  $D_1$  and second from  $D_2$ .
- ◆ Any subset of Cartesian product is a relation; e.g.  
 $R = \{(2, 1), (4, 1)\}$
- ◆ May specify which pairs are in relation using some condition for selection; e.g.

– second element is 1:

$$R = \{(x, y) \mid x \in D_1, y \in D_2, \text{ and } y = 1\}$$

– first element is always twice the second:

$$S = \{(x, v) \mid x \in D_1, v \in D_2, \text{ and } x = 2v\}$$

## Data Base Management Systems

- ◆ Consider three sets  $D_1, D_2, D_3$  with Cartesian Product  $D_1 \times D_2 \times D_3$ ; e.g.

$$D_1 = \{1, 3\} \quad D_2 = \{2, 4\} \quad D_3 = \{5, 6\}$$

$$D_1 \times D_2 \times D_3 = \{(1,2,5), (1,2,6), (1,4,5), (1,4,6), (3,2,5), (3,2,6), (3,4,5), (3,4,6)\}$$

- ◆ Any subset of these ordered triples is a relation.
- ◆ The Cartesian product of  $n$  sets  $(D_1, D_2, \dots, D_n)$  is:

$$D_1 \times D_2 \times \dots \times D_n = \{(d_1, d_2, \dots, d_n) \mid d_1 \in D_1, d_2 \in D_2, \dots, d_n \in D_n\}$$

usually written as:

$$\prod_{i=1}^n D_i$$

- ◆ Any set of  $n$ -tuples from this Cartesian product is a relation on the  $n$  sets.

### Database Relations

- ◆ Relation schema
  - Named relation defined by a set of attribute and domain name pairs.
- ◆ Relational database schema
  - Set of relation schemas, each with a distinct name.

### Properties of Relations

- ◆ Relation name is distinct from all other relation names in relational schema.
- ◆ Each cell of relation contains exactly one atomic (single) value.
- ◆ Each attribute has a distinct name.
- ◆ Values of an attribute are all from the same domain.
- ◆ Each tuple is distinct; there are no duplicate tuples.
- ◆ Order of attributes has no significance.
- ◆ Order of tuples has no significance, theoretically.

### Relational Keys

- ◆ **Superkey**
  - An attribute, or a set of attributes, that uniquely identifies a tuple within a relation.

- ◆ **Candidate Key**

- Superkey (K) such that no proper subset is a superkey within the relation.

## Data Base Management Systems

- In each tuple of R, values of K uniquely identify that tuple (uniqueness).
- No proper subset of K has the uniqueness property (irreducibility).

### ◆ **Primary Key**

- Candidate key selected to identify tuples uniquely within relation.

### ◆ **Alternate Keys**

- Candidate keys that are not selected to be primary key.

### ◆ **Foreign Key**

- Attribute, or set of attributes, within one relation that matches candidate key of some (possibly same) relation.

## Relational Integrity

### ◆ Null

- Represents value for an attribute that is currently unknown or not applicable for tuple.
- Deals with incomplete or exceptional data.
- Represents the absence of a value and is not the same as zero or spaces, which are values.

### ◆ Entity Integrity

- In a base relation, no attribute of a primary key can be null.

### ◆ Referential Integrity

- If foreign key exists in a relation, either foreign key value must match a candidate key value of some tuple in its home relation or foreign key value must be wholly null.

### ◆ **Enterprise Constraints**

- Additional rules specified by users or database administrators.

## Views

### ◆ Base Relation

- Named relation corresponding to an entity in conceptual schema, whose tuples are physically stored in database.

### ◆ View

- Dynamic result of one or more relational operations operating on base relations to produce another relation.

- ◆ A virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.

## Data Base Management Systems

- ◆ Contents of a view are defined as a query on one or more base relations.
- ◆ Views are dynamic, meaning that changes made to base relations that affect view attributes are immediately reflected in the view.

### Purpose of Views

- ◆ Provides powerful and flexible security mechanism by hiding parts of database from certain users.
- ◆ Permits users to access data in a customized way, so that same data can be seen by different users in different ways, at same time.
- ◆ Can simplify complex operations on base relations.

### Updating Views

- ◆ All updates to a base relation should be immediately reflected in all views that reference that base relation.
- ◆ If view is updated, underlying base relation should reflect change.
- ◆ There are restrictions on types of modifications that can be made through views:
  - ◆ Updates are allowed if query involves a single base relation and contains a candidate key of base relation.
  - ◆ Updates are not allowed involving multiple base relations.
  - ◆ Updates are not allowed involving aggregation or grouping operations.
- ◆ Classes of views are defined as:
  - theoretically not updateable;
  - theoretically updateable;
  - partially updateable.

## Data Base Management Systems

### Chapter 05

#### Query Languages: Relational Algebra :

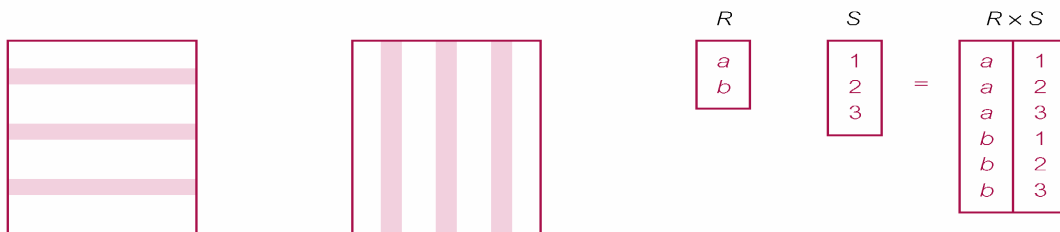
##### Introduction

- ◆ Relational algebra and relational calculus are formal languages associated with the relational model.
- ◆ Informally, relational algebra is a (high-level) procedural language and relational calculus a non-procedural language.
- ◆ However, formally both are equivalent to one another.
- ◆ A language that produces a relation that can be derived using relational calculus is relationally complete.

##### Relational Algebra

- ◆ Relational algebra operations work on one or more relations to define another relation without changing the original relations.
- ◆ Both operands and results are relations, so output from one operation can become input to another operation.
- ◆ Allows expressions to be nested, just as in arithmetic. This property is called **closure**.
- ◆ Five basic operations in relational algebra: Selection, Projection, Cartesian product, Union, and Set Difference.
- ◆ These perform most of the data retrieval operations needed.
- ◆ Also have Join, Intersection, and Division operations, which can be expressed in terms of 5 basic operations.

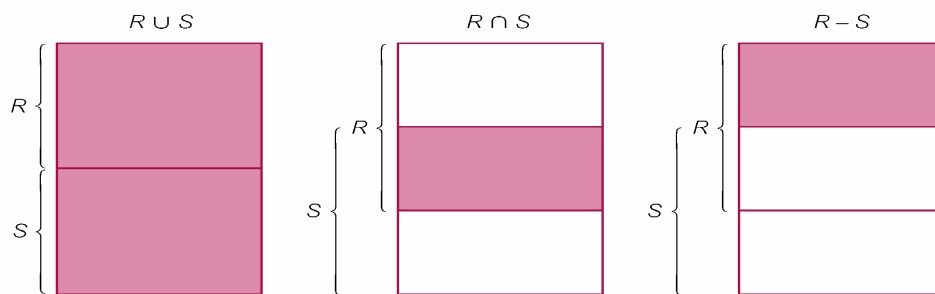
##### Relational Algebra Operations



(a) Selection

(b) Projection

(c) Cartesian product

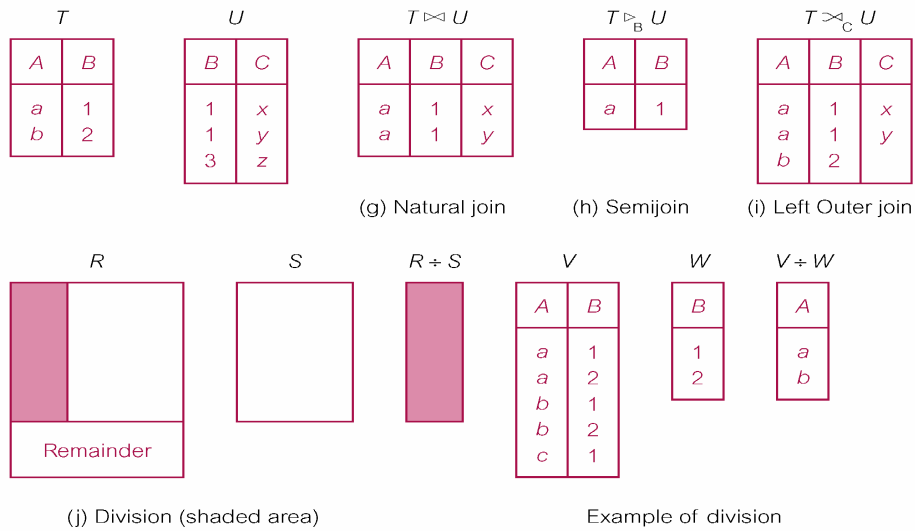


(d) Union

(e) Intersection

(f) Set difference

## Data Base Management Systems



### Selection (or Restriction)

◆  $\sigma_{\text{predicate}}(R)$

- Works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (*predicate*).

### Example - Selection (or Restriction)

- ◆ List all staff with a salary greater than £10,000.

$\sigma_{\text{salary} > 10000}(\text{Staff})$

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

### Projection

◆  $\Pi_{\text{col}_1, \dots, \text{col}_n}(R)$

- Works on a single relation R and defines a relation that contains a vertical subset of R, extracting the values of specified attributes and eliminating duplicates.

## Data Base Management Systems

### Example - Projection

- ◆ Produce a list of salaries for all staff, showing only staffNo, fName, lName, and salary details.

$\Pi_{\text{staffNo, fName, lName, salary}}(\text{Staff})$

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

### Union

- ◆  $R \cup S$ 
  - Union of two relations R and S defines a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated.
  - R and S must be union-compatible.
- ◆ If R and S have I and J tuples, respectively, union is obtained by concatenating them into one relation with a maximum of (I + J) tuples.

### Example - Union

- ◆ List all cities where there is either a branch office or a property for rent.

$\Pi_{\text{city}}(\text{Branch}) \cup \Pi_{\text{city}}(\text{PropertyForRent})$

city
London
Aberdeen
Glasgow
Bristol



## Data Base Management Systems

### Set Difference

- ◆  $R - S$ 
  - Defines a relation consisting of the tuples that are in relation R, but not in S.
  - R and S must be union-compatible.

### Example - Set Difference

- ◆ List all cities where there is a branch office but no properties for rent.  
 $\Pi_{\text{city}}(\text{Branch}) - \Pi_{\text{city}}(\text{PropertyForRent})$

city
Bristol

### Intersection

- ◆  $R \cap S$ 
  - Defines a relation consisting of the set of all tuples that are in both R and S.
  - R and S must be union-compatible.
- ◆ Expressed using basic operations:  
 $R \cap S = R - (R - S)$

### Example - Intersection

- ◆ List all cities where there is both a branch office and at least one property for rent.

$$\Pi_{\text{city}}(\text{Branch}) \cap \Pi_{\text{city}}(\text{PropertyForRent})$$

city
Aberdeen
London
Glasgow

### Cartesian product

- ◆  $R \times S$ 
  - Defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.

## Data Base Management Systems

### Example - Cartesian product

- ◆ List the names and comments of all clients who have viewed a property for rent.

$(\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \times (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR56	PA14	too small
CR76	John	Kay	CR76	PG4	too remote
CR76	John	Kay	CR56	PG4	
CR76	John	Kay	CR62	PA14	no dining room
CR76	John	Kay	CR56	PG36	
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR62	PA14	no dining room
CR56	Aline	Stewart	CR56	PG36	
CR74	Mike	Ritchie	CR56	PA14	too small
CR74	Mike	Ritchie	CR76	PG4	too remote
CR74	Mike	Ritchie	CR56	PG4	
CR74	Mike	Ritchie	CR62	PA14	no dining room
CR74	Mike	Ritchie	CR56	PG36	
CR62	Mary	Tregear	CR56	PA14	too small
CR62	Mary	Tregear	CR76	PG4	too remote
CR62	Mary	Tregear	CR56	PG4	
CR62	Mary	Tregear	CR62	PA14	no dining room
CR62	Mary	Tregear	CR56	PG36	

### Example - Cartesian product and Selection

- ◆ Use selection operation to extract those tuples where Client.clientNo = Viewing.clientNo.

$\sigma_{\text{Client.clientNo} = \text{Viewing.clientNo}}((\Pi_{\text{clientNo, fName, lName}}(\text{Client})) \times (\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing})))$

- ◆ Cartesian product and Selection can be reduced to a single operation called a *Join*.

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	no dining room

### Join Operations

- ◆ Join is a derivative of Cartesian product.

## Data Base Management Systems

- ◆ Equivalent to performing a Selection, using join predicate as selection formula, over Cartesian product of the two operand relations.
- ◆ One of the most difficult operations to implement efficiently in an RDBMS and one reason why RDBMSs have intrinsic performance problems.
- ◆ **Various forms of join operation**
  - ◆ Theta join
  - ◆ Equijoin (a particular type of Theta join)
  - ◆ Natural join
  - ◆ Outer join
  - ◆ Semijoin

### Theta join ( $\theta$ -join)

- ◆  $R \bowtie_{FS} S$ 
  - Defines a relation that contains tuples satisfying the predicate F from the Cartesian product of R and S.
  - The predicate F is of the form  $R.a_i \theta S.b_i$  where  $\theta$  may be one of the comparison operators ( $<, \leq, >, \geq, =, \neq$ ).
- ◆ Can rewrite Theta join using basic Selection and Cartesian product operations.  
 $R \bowtie_{FS} S = \sigma_F(R \times S)$
- ◆ Degree of a Theta join is sum of degrees of the operand relations R and S. If predicate F contains only equality ( $=$ ), the term *Equijoin* is used.

### Example - Equijoin

- ◆ List the names and comments of all clients who have viewed a property for rent.

$(\Pi_{clientNo, fName, lName}(Client)) \bowtie_{Client.clientNo = Viewing.clientNo}$   
 $(\Pi_{clientNo, propertyNo, comment}(Viewing))$

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR56	PG36	
CR62	Mary	Tregear	CR62	PA14	no dining room

## Data Base Management Systems

### Natural join

- ◆ R S
  - An Equijoin of the two relations R and S over all common attributes x. One occurrence of each common attribute is eliminated from the result.

### Example - Natural join

- ◆ List the names and comments of all clients who have viewed a property for rent.

$(\Pi_{\text{clientNo, fName, lName}}(\text{Client}))$   
 $(\Pi_{\text{clientNo, propertyNo, comment}}(\text{Viewing}))$

clientNo	fName	lName	propertyNo	comment
CR76	John	Kay	PG4	too remote
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR56	Aline	Stewart	PG36	
CR62	Mary	Tregear	PA14	no dining room

### Outer join

- ◆ To display rows in the result that do not have matching values in the join column, use Outer join.
- ◆ R S
  - (Left) outer join is join in which tuples from R that do not have matching values in common columns of S are also included in result relation.

### Example - Left Outer join

- ◆ Produce a status report on property viewings.

$\Pi_{\text{propertyNo, street, city}}(\text{PropertyForRent})$   
 $\text{Viewing}$

propertyNo	street	city	clientNo	viewDate	comment
PA14	16 Holhead	Aberdeen	CR56	24-May-01	too small
PA14	16 Holhead	Aberdeen	CR62	14-May-01	no dining room
PL94	6 Argyll St	London	null	null	null
PG4	6 Lawrence St	Glasgow	CR76	20-Apr-01	too remote
PG4	6 Lawrence St	Glasgow	CR56	26-May-01	
PG36	2 Manor Rd	Glasgow	CR56	28-Apr-01	
PG21	18 Dale Rd	Glasgow	null	null	null
PG16	5 Novar Dr	Glasgow	null	null	null

$\Pi_{\text{clientNo, propertyNo}}(\text{Viewing})$ 

clientNo	propertyNo
CR56	PA14
CR76	PG4
CR56	PG4
CR62	PA14
CR56	PG36

 $\Pi_{\text{propertyNo}}(\sigma_{\text{rooms}=3}(\text{PropertyForRent}))$ 

propertyNo
PG4
PG36

RESULT

clientNo
CR56

### ◆ $R \bowtie_{FS}$

- Defines a relation that contains the tuples of R that participate in the join of R with S.
- Can rewrite Semijoin using Projection and Join:
- $R \bowtie_{FS} S = \Pi_A(R \bowtie_{FS} S)$

### Example - Semijoin

- ◆ List complete details of all staff who work at the branch in Glasgow.

Staff  $\bowtie_{\text{Staff.branchNo} = \text{Branch.branchNo and Branch.city} = \text{'Glasgow'}} \text{Branch}$

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

### Division

#### ◆ $R \div S$

- Defines a relation over the attributes C that consists of set of tuples from R that match combination of every tuple in S.

- ◆ Expressed using basic operations:

$$T1 \leftarrow \Pi_C(R)$$

$$T2 \leftarrow \Pi_C((S \times T1) - R)$$

$$T \leftarrow T1 - T2$$

### Example - Division

- ◆ Identify all clients who have viewed all properties with three rooms.

$$(\Pi_{\text{clientNo, propertyNo}}(\text{Viewing})) \div (\Pi_{\text{propertyNo}}(\sigma_{\text{rooms} = 3}(\text{PropertyForRent})))$$

## Data Base Management Systems

### Relational Calculus

- ◆ Relational calculus query specifies *what* is to be retrieved rather than *how* to retrieve it.
  - No description of how to evaluate a query.
- ◆ In first-order logic (or predicate calculus), *predicate* is a truth-valued function with arguments.
- ◆ When we substitute values for the arguments, function yields an expression, called a *proposition*, which can be either true or false.
- ◆ If predicate contains a variable (e.g. 'x is a member of staff'), there must be a range for x.
- ◆ When we substitute some values of this range for x, proposition may be true; for other values, it may be false.
- ◆ When applied to databases, relational calculus has forms: *tuple* and *domain*.

### Tuple Relational Calculus

- ◆ Interested in finding tuples for which a predicate is true. Based on use of tuple variables.
- ◆ Tuple variable is a variable that 'ranges over' a named relation: i.e., variable whose only permitted values are tuples of the relation.
- ◆ Specify range of a tuple variable *S* as the Staff relation as:  
Staff(*S*)
- ◆ To find set of all tuples *S* such that *P(S)* is true:  
{*S* | *P(S)*}

### **Tuple Relational Calculus - Example**

- ◆ To find details of all staff earning more than £10,000:  
{*S* | Staff(*S*) ∧ *S*.salary > 10000}
- ◆ To find a particular attribute, such as salary, write:  
{*S*.salary | Staff(*S*) ∧ *S*.salary > 10000}

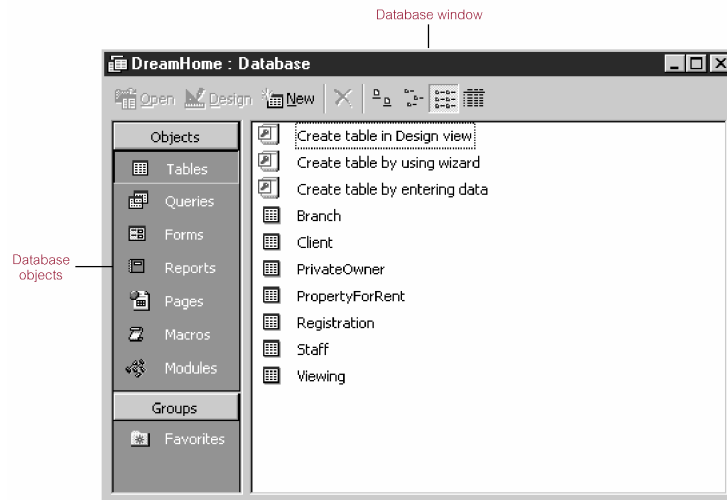
### Query-by-Example (QBE)

- ◆ Visual approach for accessing information in a database through use of query templates.
- ◆ Example values are entered into template to represent what access to database is to achieve, such as the answer to a query.
- ◆ Originally developed by IBM in 1970s and has proved so popular that QBE (or similar) is now provided by most DBMSs.
- ◆ When user constructs a QBE - in background, DBMS creates an equivalent SQL statement.

## Data Base Management Systems

- Ask questions about data in one or more tables.
- Specify the fields we want in the answer.
- Select records according to some criteria.
- Perform calculations on the data in tables.
- Insert and delete records.
- Modify values of fields.
- Create new fields and tables

### Introduction to Microsoft Access



### Summary of Microsoft Access Query Types

**Table 7.1** Summary of Microsoft Access 2000 query types.

Query type	Description
Select query	Asks a question or defines a set of criteria about the data in one or more tables.
Totals (Aggregate) query	Performs calculations on groups of records.
Parameter query	Displays one or more predefined dialog boxes that prompts the user for the parameter value(s).
Find Matched query	Finds duplicate records in a single table.
Find Unmatched query	Finds distinct records in related tables.
Crosstab query	Allows large amounts of data to be summarized and presented in a compact spreadsheet.
Autolookup query	Automatically fills in certain field values for a new record.
Action query (including delete, append, update, and make-table queries)	Makes changes to many records in just one operation. Such changes include the ability to delete, append, or make changes to records in a table and also to create a new table.
SQL query (including union, pass-through, data definition, and subqueries)	Used to modify the queries described above and to set the properties of forms and reports. Must be used to create SQL-specific queries such as union, data definition, subqueries (see Chapters 5 and 6), and pass-through queries. Pass-through queries send commands to a SQL database such as Microsoft or Sybase SQL Server.

## Data Base Management Systems

### Introduction to Microsoft Access Queries



### Building Select Queries using QBE - Specifying Criteria

(a)

PropertyForRent  
field list

QBE  
grid

Selected **propertyNo**, **city**, **type**, and **rent** fields displayed as columns

(b)

Datasheet

(c)

```
SELECT PropertyForRent.propertyNo, PropertyForRent.city, PropertyForRent.type, PropertyForRent.rent
FROM PropertyForRent;
```



## Data Base Management Systems

(a)

QBE grid

Field:	propertyNo	city	type	rent
Table:	PropertyForRent	PropertyForRent	PropertyForRent	PropertyForRent
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:		"Glasgow"		Between 350 And 450
or:				

Criteria on same row so combined using *And* operator

Criteria using *And* operator

(b)

Datasheet

Query1 : Select Query				
propertyNo	city	type	rent	
PG4	Glasgow	Flat	350	
PG36	Glasgow	Flat	375	
PG16	Glasgow	Flat	450	

Records that satisfy criteria

(c)

```

SELECT PropertyForRent.propertyNo, PropertyForRent.city, PropertyForRent.type, PropertyForRent.rent
FROM PropertyForRent
WHERE (((PropertyForRent.city)="Glasgow") AND ((PropertyForRent.rent) Between 350 And 450));
    
```

---

(a)

QBE grid

Field:	propertyNo	city	type	rent
Table:	PropertyForRent	PropertyForRent	PropertyForRent	PropertyForRent
Sort:				
Show:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Criteria:		"Glasgow"		Between 350 And 450
or:		"Aberdeen"		

Criteria on different rows so combined using *Or* operator

Criteria on same row so combined using *And* operator

(b)

Datasheet

Query1 : Select Query				
propertyNo	city	type	rent	
PA14	Aberdeen	House	650	
PG4	Glasgow	Flat	350	
PG36	Glasgow	Flat	375	
PG16	Glasgow	Flat	450	

Records that satisfy criteria

(c)

```

SELECT PropertyForRent.propertyNo, PropertyForRent.city, PropertyForRent.type, PropertyForRent.rent
FROM PropertyForRent
WHERE (((PropertyForRent.city)="Glasgow") AND ((PropertyForRent.rent) Between 350 And 450)) OR
(((PropertyForRent.city)="Aberdeen"));
    
```

---

## Data Base Management Systems

### Chapter : 06

#### SQL

##### Objectives of SQL

- ◆ Ideally, database language should allow user to:
  - create the database and relation structures;
  - perform insertion, modification, deletion of data from relations;
  - perform simple and complex queries.
- ◆ Must perform these tasks with minimal user effort and command structure/syntax must be easy to learn.
- ◆ It must be portable.
- ◆ SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.
- ◆ Until SQL3, SQL did not contain flow of control commands. These had to be implemented using a programming or job-control language, or interactively by the decisions of user.
- ◆ SQL is relatively easy to learn:
  - it is non-procedural - you specify *what* information you require, rather than *how* to get it;
  - it is essentially free-format.
- ◆ Consists of standard English words:
  - 1) CREATE TABLE Staff(staffNo VARCHAR(5),  
                          lName VARCHAR(15),  
                          salary DECIMAL(7,2));
  - 2) INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
  - 3) SELECT staffNo, lName, salary  
      FROM Staff  
      WHERE salary > 10000;
- ◆ Can be used by range of users including DBAs, management, application developers, and other types of end users.
- ◆ An ISO standard now exists for SQL, making it both the formal and *de facto* standard language for relational databases.

## Data Base Management Systems

### History of SQL

- ◆ In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' (SEQUEL).
- ◆ A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
- ◆ Still pronounced 'see-quel', though official pronunciation is 'S-Q-L'.
- ◆ IBM subsequently produced a prototype DBMS called *System R*, based on SEQUEL/2.
- ◆ Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.
- ◆ In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.
- ◆ In 1987, ANSI and ISO published an initial standard for SQL.
- ◆ In 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.
- ◆ In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
- ◆ In 1999, SQL3 was released with support for object-oriented data management.

### Importance of SQL

- ◆ SQL has become part of application architectures such as IBM's Systems Application Architecture.
- ◆ It is strategic choice of many large and influential organizations (e.g. X/OPEN).
- ◆ SQL is Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to American Government.
- ◆ SQL is used in other standards and even influences development of other standards as a definitional tool. Examples include:
  - ISO's Information Resource Directory System (IRDS) Standard
  - Remote Data Access (RDA) Standard.

### Writing SQL Commands

- ◆ SQL statement consists of **reserved words** and **user-defined words**.

## Data Base Management Systems

- Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
- User-defined words are made up by user and represent names of various database objects such as relations, columns, views.
  
- ◆ Most components of an SQL statement are *case insensitive*, except for literal character data.
- ◆ More readable with indentation and lineation:
  - Each clause should begin on a new line.
  - Start of a clause should line up with start of other clauses.
  - If clause has several parts, should each appear on a separate line and be indented under start of clause.
  
- ◆ Use extended form of BNF notation:
  - Upper-case letters represent reserved words.
  - Lower-case letters represent user-defined words.
  - | indicates a *choice* among alternatives.
  - Curly braces indicate a *required element*.
  - Square brackets indicate an *optional element*.
  - ... indicates *optional repetition* (0 or more).

### Literals

- ◆ Literals are constants used in SQL statements.
- ◆ All non-numeric literals must be enclosed in single quotes (e.g. 'London').
- ◆ All numeric literals must not be enclosed in quotes (e.g. 650.00).

## Data Base Management Systems

# SQL Statements

SELECT	Data retrieval
INSERT UPDATE DELETE MERGE	Data manipulation language (DML)
CREATE ALTER DROP RENAME TRUNCATE	Data definition language (DDL)
COMMIT ROLLBACK SAVEPOINT	Transaction control
GRANT REVOKE	Data control language (DCL)

### SELECT Statement

```
SELECT [DISTINCT | ALL]
      { * | [columnExpression [AS newName]] [, ...] }
FROM   TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList]
```

FROM Specifies table(s) to be used.

WHERE Filters rows.

GROUP BY Forms groups of rows with same column value.

HAVING Filters groups subject to some condition.

SELECT Specifies which columns are to appear in output.

ORDER BY Specifies the order of the output.

- ◆ Order of the clauses cannot be changed.
- ◆ Only SELECT and FROM are mandatory.

### Example 5.1 All Columns, All Rows

List full details of all staff.

```
SELECT staffNo, fName, lName, address,
       position, salary, branchNo
```

## Data Base Management Systems

```
FROM Staff;
```

◆ Can use \* as an abbreviation for 'all columns':

```
SELECT *
FROM Staff;
```

**Table 5.1** Result table for Example 5.1.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

### Specific Columns, All Rows

Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary
FROM Staff;
```

**Table 5.2** Result table for Example 5.2.

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

### Use of DISTINCT

List the property numbers of all properties that have been viewed.

```
SELECT propertyNo
FROM Viewing;
```

## Data Base Management Systems

propertyNo
PA14
PG4
PG4
PA14
PG36

- ◆ Use DISTINCT to eliminate duplicates:  

```
SELECT DISTINCT propertyNo
FROM Viewing;
```

propertyNo
PA14
PG4
PG36

### Calculated Fields

Produce a list of monthly salaries for all staff, showing staff number, first and last names, and salary details.

```
SELECT staffNo, fName, lName, salary/12
FROM Staff;
```

**Table 5.4** Result table for Example 5.4.

staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

- ◆ To name column, use AS clause:

```
SELECT staffNo, fName, lName, salary/12
      AS monthlySalary
FROM Staff;
```

### Comparison Search Condition

List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000;
```

## Data Base Management Systems

**Table 5.5** Result table for Example 5.5.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

### Compound Comparison Search Condition

List addresses of all branch offices in London or Glasgow.

```
SELECT *
FROM Branch
WHERE city = 'London' OR city = 'Glasgow';
```

**Table 5.6** Result table for Example 5.6.

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

### Range Search Condition

List all staff with a salary between 20,000 and 30,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

- ◆ BETWEEN test includes the endpoints of range.

**Table 5.7** Result table for Example 5.7.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

- ◆ Also a negated version NOT BETWEEN.
- ◆ BETWEEN does not add much to SQL's expressive power.  
Could also write:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
```



## Data Base Management Systems

```
WHERE salary >= 20000 AND salary <= 30000;
```

- ◆ Useful, though, for a range of values.

### Set Membership

List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

- ◆ There is a negated version (NOT IN).
- ◆ IN does not add much to SQL's expressive power.
- ◆ Could have expressed this as:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position = 'Manager' OR
       position = 'Supervisor';
```

- ◆ IN is more efficient when set contains many values

### Pattern Matching

Find all owners with the string 'Glasgow' in their address.

```
SELECT clientNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE '%Glasgow%';
```

**Table 5.9** Result table for Example 5.9.

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

- ◆ SQL has two special pattern matching symbols:
  - %: sequence of zero or more characters;
  - \_ (underscore): any single character.
- ◆ LIKE '%Glasgow%' means a sequence of characters of any length containing 'Glasgow'.

### NULL Search Condition

List details of all viewings on property PG4 where a comment has not been supplied.

- ◆ There are 2 viewings for property PG4, one with and one without a comment.

## Data Base Management Systems

- ◆ Have to test for null explicitly using special keyword IS NULL:

```
SELECT clientNo, viewDate
FROM Viewing
WHERE propertyNo = 'PG4' AND
      comment IS NULL;
```

- ◆ Negated version (IS NOT NULL) can test for non-null values.

### Single Column Ordering

List salaries for all staff, arranged in descending order of salary.

```
SELECT staffNo, fName, lName, salary
FROM Staff
ORDER BY salary DESC;
```

**Table 5.11** Result table for Example 5.11.

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

### Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type;
```

## Data Base Management Systems

**Table 5.12(a)** Result table for Example 5.12 with one sort key.

propertyNo	type	rooms	rent
PL94	Flat	4	400
PG4	Flat	3	350
PG36	Flat	3	375
PG16	Flat	4	450
PA14	House	6	650
PG21	House	5	600

- ◆ Four flats in this list - as no minor sort key specified, system arranges these rows in any order it chooses.
- ◆ To arrange in order of rent, specify minor order:

```
SELECT propertyNo, type, rooms, rent
FROM PropertyForRent
ORDER BY type, rent DESC;
```

### SELECT Statement - Aggregates

- ◆ ISO standard defines five aggregate functions:

COUNT returns number of values in specified column.  
 SUM returns sum of values in specified column.  
 AVG returns average of values in specified column.  
 MIN returns smallest value in specified column.  
 MAX returns largest value in specified column.

- ◆ Each operates on a single column of a table and returns a single value.
- ◆ COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- ◆ Apart from COUNT(\*), each function eliminates nulls first and operates only on remaining non-null values.
- ◆ COUNT(\*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- ◆ Can use DISTINCT before column name to eliminate duplicates.
- ◆ DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.
- ◆ Aggregate functions can be used only in SELECT list and in HAVING clause.
- ◆ If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column out with an aggregate function. For example, the following is illegal:

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

## Data Base Management Systems

### Use of COUNT (DISTINCT)

How many different properties viewed in May '01?

```
SELECT COUNT(DISTINCT propertyNo) AS count
FROM Viewing
WHERE viewDate BETWEEN '1-May-01'
      AND '31-May-01';
```

### Use of COUNT and SUM

Find number of Managers and sum of their salaries.

```
SELECT COUNT(staffNo) AS count,
      SUM(salary) AS sum
FROM Staff
WHERE position = 'Manager';
```

**Table 5.15** Result table for Example 5.15.

count	sum
2	54000.00

### Use of MIN, MAX, AVG

Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS min,
      MAX(salary) AS max,
      AVG(salary) AS avg
FROM Staff;
```

**Table 5.16** Result table for Example 5.16.

min	max	avg
9000.00	30000.00	17000.00

### SELECT Statement - Grouping

- ◆ Use GROUP BY clause to get sub-totals.
- ◆ SELECT and GROUP BY closely integrated: each item in SELECT list must be *single-valued* per group, and SELECT

## Data Base Management Systems

- column names
- aggregate functions
- constants
- expression involving combinations of the above.

- ◆ All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- ◆ If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ◆ ISO considers two nulls to be equal for purposes of GROUP BY.

Find number of staff in each branch and their total salaries.

```
SELECT    branchNo,
          COUNT(staffNo) AS count,
          SUM(salary) AS sum
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

### Restricted Groupings - HAVING clause

- ◆ HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
- ◆ Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- ◆ Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT branchNo,
          COUNT(staffNo) AS count,
          SUM(salary) AS sum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

### Subqueries

- ◆ Some SQL statements can have a SELECT embedded within them.
- ◆ A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a *subquery* or *nested query*.
- ◆ Subselects may also appear in INSERT, UPDATE, and DELETE statements.

## Data Base Management Systems

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo =
      (SELECT branchNo
       FROM Branch
       WHERE street = '163 Main St');
```

### Subquery with Equality

- ◆ Inner SELECT finds branch number for branch at '163 Main St' ('B003').
- ◆ Outer SELECT then retrieves details of all staff who work at this branch.
- ◆ Outer SELECT then becomes:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = 'B003';
```

**Table 5.19** Result table for Example 5.19.

staffNo	fName	lName	position
SG37	Ann	Beech	Assistant
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

### Subquery with Aggregate

List all staff whose salary is greater than the average salary, and show by how much.

```
SELECT staffNo, fName, lName, position,
       salary - (SELECT AVG(salary) FROM Staff) As SalDiff
FROM Staff
WHERE salary >
      (SELECT AVG(salary)
       FROM Staff);
```

- ◆ Cannot write 'WHERE salary > AVG(salary)'
- ◆ Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

## Data Base Management Systems

```

        salary - 17000 As salDiff
FROM Staff
WHERE salary > 17000;

```

### Subquery Rules

- ◆ ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).
- ◆ Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.
- ◆ By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.
- ◆ When subquery is an operand in a comparison, subquery must appear on right-hand side.
- ◆ A subquery may not be used as an operand in an expression.

### Nested subquery: use of IN

List properties handled by staff at '163 Main St'.

```

SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
WHERE staffNo IN
  (SELECT staffNo
   FROM Staff
   WHERE branchNo =
     (SELECT branchNo
      FROM Branch
      WHERE street = '163 Main St'));

```

### Multi-Table Queries

- ◆ Can use subqueries provided result columns come from same table.
- ◆ If result columns come from more than one table must use a join.
- ◆ To perform join, include more than one table in FROM clause.

## Data Base Management Systems

- ◆ Use comma as separator and typically include WHERE clause to specify join column(s).
- ◆ Also possible to use an alias for a table named in FROM clause.
- ◆ Alias is separated from table name with a space.
- ◆ Alias can be used to qualify column names when there is ambiguity.

### Simple Join

List names of all clients who have viewed a property along with any comment supplied.

```
SELECT c.clientNo, fName, lName,
       propertyNo, comment
FROM Client c, Viewing v
WHERE c.clientNo = v.clientNo;
```

- ◆ Only those rows from both tables that have identical values in the clientNo columns (c.clientNo = v.clientNo) are included in result.
- ◆ Equivalent to equi-join in relational algebra.

**Table 5.24** Result table for Example 5.24.

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

### Alternative JOIN Constructs

- ◆ SQL provides alternative ways to specify joins:

```
FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo
FROM Client JOIN Viewing USING clientNo
FROM Client NATURAL JOIN Viewing
```



## Data Base Management Systems

- ◆ In each case, FROM replaces original FROM and WHERE. However, first produces table with two identical clientNo columns.

### Sorting a join

For each branch, list numbers and names of staff who manage properties, and properties they manage.

```
SELECT s.branchNo, s.staffNo, fName, lName,
       propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, propertyNo;
```

**Table 5.25** Result table for Example 5.25.

branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

### Three Table Join

For each branch, list staff who manage properties, including city in which branch is located and properties they manage.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName,
       propertyNo
FROM Branch b, Staff s, PropertyForRent p
WHERE b.branchNo = s.branchNo AND
       s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo
```

**Table 5.26** Result table for Example 5.26.

branchNo	city	staffNo	fName	lName	propertyNo
B003	Glasgow	SG14	David	Ford	PG16
B003	Glasgow	SG37	Ann	Beech	PG21
B003	Glasgow	SG37	Ann	Beech	PG36
B005	London	SL41	Julie	Lee	PL94
B007	Aberdeen	SA9	Mary	Howe	PA14

- ◆ Alternative formulation for FROM and WHERE:

```
FROM (Branch b JOIN Staff s USING branchNo) AS
      bs JOIN PropertyForRent p USING staffNo
```

## Data Base Management Systems

### Computing a Join

Procedure for generating results of a join are:

1. Form Cartesian product of the tables named in FROM clause.
2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.
3. For each remaining row, determine value of each item in SELECT list to produce a single row in result table.
4. If DISTINCT has been specified, eliminate any duplicate rows from the result table.
5. If there is an ORDER BY clause, sort result table as required.

- ◆ SQL provides special format of SELECT for Cartesian product:

```
SELECT [DISTINCT | ALL] { * | columnList }
FROM Table1 CROSS JOIN Table2
```

### Outer Joins

- ◆ If one row of a joined table is unmatched, row is omitted from result table.
- ◆ Outer join operations retain rows that do not satisfy the join condition.
- ◆ Consider following tables:

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

- ◆ Result table has two rows where cities are same.
- ◆ There are no rows corresponding to branches in Bristol and Aberdeen.
- ◆ To include unmatched rows in result table, use an Outer

## Data Base Management Systems

### Left Outer Join

List branches and properties that are in same city along with any unmatched branches.

```
SELECT b.*, p.*
FROM Branch1 b LEFT JOIN
      PropertyForRent1 p ON b.bCity = p.pCity;
```

- ◆ Includes those rows of first (left) table unmatched with rows from second (right) table.
- ◆ Columns from second table are filled with NULLs.

**Table 5.28** Result table for Example 5.28.

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

### Right Outer Join

List branches and properties in same city and any unmatched properties.

```
SELECT b.*, p.*
FROM Branch1 b RIGHT JOIN
      PropertyForRent1 p ON b.bCity = p.pCity;
```

- ◆ Right Outer join includes those rows of second (right) table that are unmatched with rows from first (left) table.
- ◆ Columns from first table are filled with NULLs.

**Table 5.29** Result table for Example 5.29.

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

## Data Base Management Systems

### Full Outer Join

List branches and properties in same city and any unmatched branches or properties.

```
SELECT b.*, p.*
FROM Branch1 b FULL JOIN
PropertyForRent1 p ON b.bCity = p.pCity;
```

- ◆ Includes rows that are unmatched in both tables.
- ◆ Unmatched columns are filled with NULLs.

### **Integrity Enhancement Feature**

- ◆ Consider five types of integrity constraints:
  - Required data.
  - Domain constraints.
  - Entity integrity.
  - Referential integrity.
  - Enterprise constraints.

### **Integrity Enhancement Feature**

#### Required Data

```
position VARCHAR(10) NOT NULL
```

#### Domain Constraints

##### (a) CHECK

```
sex CHAR NOT NULL
CHECK (sex IN ('M', 'F'))
```

#### CREATE DOMAIN

```
CREATE DOMAIN DomainName [AS] dataType
[DEFAULT defaultOption]
[CHECK (searchCondition)]
```

For example:

```
CREATE DOMAIN SexType AS CHAR
CHECK (VALUE IN ('M', 'F'));
sex SexType NOT NULL
```

- ◆ *searchCondition* can involve a table lookup:

```
CREATE DOMAIN BranchNo AS CHAR(4)
CHECK (VALUE IN (SELECT branchNo
FROM Branch));
```

## Data Base Management Systems

```
DROP DOMAIN DomainName
    [RESTRICT | CASCADE]
```

### IEF - Entity Integrity

- ◆ Primary key of a table must contain a unique, non-null value for each row.
- ◆ ISO standard supports FOREIGN KEY clause in CREATE and ALTER TABLE statements:

```
PRIMARY KEY(staffNo)
PRIMARY KEY(clientNo, propertyNo)
```

- ◆ Can only have one PRIMARY KEY clause per table. Can still ensure uniqueness for alternate keys using UNIQUE:

```
UNIQUE(telNo)
```

### IEF - Referential Integrity

- ◆ FK is column or set of columns that links each row in child table containing foreign FK to row of parent table containing matching PK.
- ◆ Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table.
- ◆ ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE:

```
FOREIGN KEY(branchNo) REFERENCES Branch
```

- ◆ Any INSERT/UPDATE that attempts to create FK value in child table without matching candidate key value in parent is rejected.
- ◆ Action taken that attempts to update/delete a candidate key value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:

```
- CASCADE                - SET NULL
- SET DEFAULT            - NO ACTION
```

CASCADE: Delete row from parent and delete matching rows in child, and so on in cascading manner.

SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are NOT NULL.

SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns

NO ACTION: Reject delete from parent. Default.

## Data Base Management Systems

FOREIGN KEY (ownerNo) REFERENCES Owner ON UPDATE CASCADE

### IEF - Enterprise Constraints

- ◆ Could use CHECK/UNIQUE in CREATE and ALTER TABLE.
- ◆ Also have:

```
CREATE ASSERTION AssertionName
CHECK (searchCondition)
```

- ◆ which is very similar to the CHECK clause.

```
CREATE ASSERTION StaffNotHandlingTooMuch
CHECK (NOT EXISTS (SELECT staffNo
                    FROM PropertyForRent
                    GROUP BY staffNo
                    HAVING COUNT(*) > 100))
```

### Data Definition

- ◆ SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.
- ◆ Main SQL DDL statements are:

```
CREATE SCHEMA          DROP SCHEMA
CREATE/ALTER DOMAIN  DROP DOMAIN
CREATE/ALTER TABLE   DROP TABLE
CREATE VIEW           DROP VIEW
```

- ◆ Many DBMSs also provide:

```
CREATE INDEX          DROP INDEX
```

- ◆ Relations and other database objects exist in an *environment*.
- ◆ Each environment contains one or more *catalogs*, and each catalog consists of set of schemas.
- ◆ Schema is named collection of related database objects.
- ◆ Objects in a schema can be tables, views, domains, assertions, collations, translations, and character sets. All have same owner.

### CREATE TABLE

```
CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
```

## Data Base Management Systems

```
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] [...,]}
{[FOREIGN KEY (listOfFKColumns)
  REFERENCES ParentTableName [(listOfCKColumns)],
  [ON UPDATE referentialAction]
  [ON DELETE referentialAction ]] [,...]}
{[CHECK (searchCondition)] [,... ]})
```

- ◆ Creates a table with one or more columns of the specified *dataType*.
- ◆ With NOT NULL, system rejects any attempt to insert a null in the column.
- ◆ Can specify a DEFAULT value for the column.
- ◆ Primary keys should always be specified as NOT NULL.
- ◆ FOREIGN KEY clause specifies FK along with the referential action

### **ALTER TABLE**

- ◆ Add a new column to a table.
- ◆ Drop a column from a table.
- ◆ Add a new table constraint.
- ◆ Drop a table constraint.
- ◆ Set a default for a column.
- ◆ Drop a default for a column.

Change Staff table by removing default of 'Assistant' for position column and setting default for sex column to female ('F').

```
ALTER TABLE Staff
  ALTER position DROP DEFAULT;
ALTER TABLE Staff
  ALTER sex SET DEFAULT 'F';
```

Remove constraint from PropertyForRent that staff not allowed to handle more than 100 properties at time. Add new column to Client table.

```
ALTER TABLE PropertyForRent
  DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
  ADD prefNoRooms PRooms;
```

### **DROP TABLE**

```
DROP TABLE TableName [RESTRICT | CASCADE]
```

## Data Base Management Systems

- ◆ Removes named table and all rows within it.
- ◆ With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- ◆ With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).

### View

Dynamic result of one or more relational operations operating on base relations to produce another relation.

- ◆ Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.
- ◆ Contents of a view are defined as a query on one or more base relations.
- ◆ With view resolution, any operations on view are automatically translated into operations on relations from which it is derived.
- ◆ With view materialization, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.

### **SQL - CREATE VIEW**

```
CREATE VIEW ViewName [ (newColumnName [,...]) ]
AS subselect
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

- ◆ Can assign a name to each column in view.
- ◆ If list of column names is specified, it must have same number of items as number of columns produced by *subselect*.
- ◆ If omitted, each column takes name of corresponding column in *subselect*.
- ◆ List must be specified if there is any ambiguity in a column name.
- ◆ The *subselect* is known as the defining query.
- ◆ WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.
- ◆ Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.

### **SQL - DROP VIEW**

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

- ◆ Causes definition of view to be deleted from database.



## Data Base Management Systems

- ◆ For example:

```
DROP VIEW Manager3Staff;
```

- ◆ With CASCADE, all related dependent objects are deleted; i.e. any views defined on view being dropped.
- ◆ With RESTRICT (default), if any other objects depend for their existence on continued existence of view being dropped, command is rejected

### Restrictions on Views

SQL imposes several restrictions on creation and use of views.

- (a) If column in view is based on an aggregate function:
- Column may appear only in SELECT and ORDER BY clauses of queries that access view.
  - Column may not be used in WHERE nor be an argument to an aggregate function in any query based on view.
- ◆ For example, following query would fail:

```
SELECT COUNT(cnt)
FROM StaffPropCnt;
```

- ◆ Similarly, following query would also fail:

```
SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

(b) Grouped view may never be joined with a base table or a view.

- ◆ For example, StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.

### Advantages of Views

- ◆ Data independence
- ◆ Currency
- ◆ Improved security
- ◆ Reduced complexity
- ◆ Convenience
- ◆ Customization
- ◆ Data integrity

### Disadvantages of Views

- ◆ Update restriction
- ◆ Structure restriction

## Data Base Management Systems

- ◆ Performance

### Transactions

- ◆ SQL defines transaction model based on COMMIT and ROLLBACK.
- ◆ Transaction is logical unit of work with one or more SQL statements guaranteed to be atomic with respect to recovery.
- ◆ An SQL transaction automatically begins with a *transaction-initiating* SQL statement (e.g., SELECT, INSERT).
- ◆ Changes made by transaction are not visible to other concurrently executing transactions until transaction completes.
- ◆ Transaction can complete in one of four ways:
  - COMMIT ends transaction successfully, making changes permanent.
  - ROLLBACK aborts transaction, backing out any changes made by transaction.
  - For programmatic SQL, successful program termination ends final transaction successfully, even if COMMIT has not been executed.
  - For programmatic SQL, abnormal program end aborts transaction.

### Access Control - Authorization Identifiers and Ownership

- ◆ Authorization identifier is normal SQL identifier used to establish identity of a user. Usually has an associated password.
- ◆ Used to determine which objects user may reference and what operations may be performed on those objects.
- ◆ Each object created in SQL has an owner, as defined in AUTHORIZATION clause of schema to which object belongs.
- ◆ Owner is only person who may know about it.

### Privileges

- ◆ Actions user permitted to carry out on given base table or view:
 

SELECT	Retrieve data from a table.
INSERT	Insert new rows into a table.
UPDATE	Modify rows of data in a table.
DELETE	Delete rows of data from a table.
REFERENCES	Reference columns of named table in integrity constraints.
USAGE	Use domains, collations, character sets, and translations.

## Data Base Management Systems

- ◆ Can restrict INSERT/UPDATE/REFERENCES to named columns.
- ◆ Owner of table must grant other users the necessary privileges using GRANT statement.
- ◆ To create view, user must have SELECT privilege on all tables that make up view and REFERENCES privilege on the named columns

### GRANT

```
GRANT      {PrivilegeList | ALL PRIVILEGES}
ON      ObjectName
TO      {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

- ◆ *PrivilegeList* consists of one or more of above privileges separated by commas.
- ◆ ALL PRIVILEGES grants all privileges to a user.
- ◆ PUBLIC allows access to be granted to all present and future authorized users.
- ◆ *ObjectName* can be a base table, view, domain, character set, collation or translation.
- ◆ WITH GRANT OPTION allows privileges to be passed on.

### Example

Give Manager full privileges to Staff table.

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager WITH GRANT OPTION;
```

Give users Personnel and Director SELECT and UPDATE on column salary of Staff.

```
GRANT SELECT, UPDATE (salary)
ON Staff
TO Personnel, Director;
```

### GRANT Specific Privileges to PUBLIC

Give all users SELECT on Branch table.

```
GRANT SELECT
ON Branch
TO PUBLIC;
```

### REVOKE

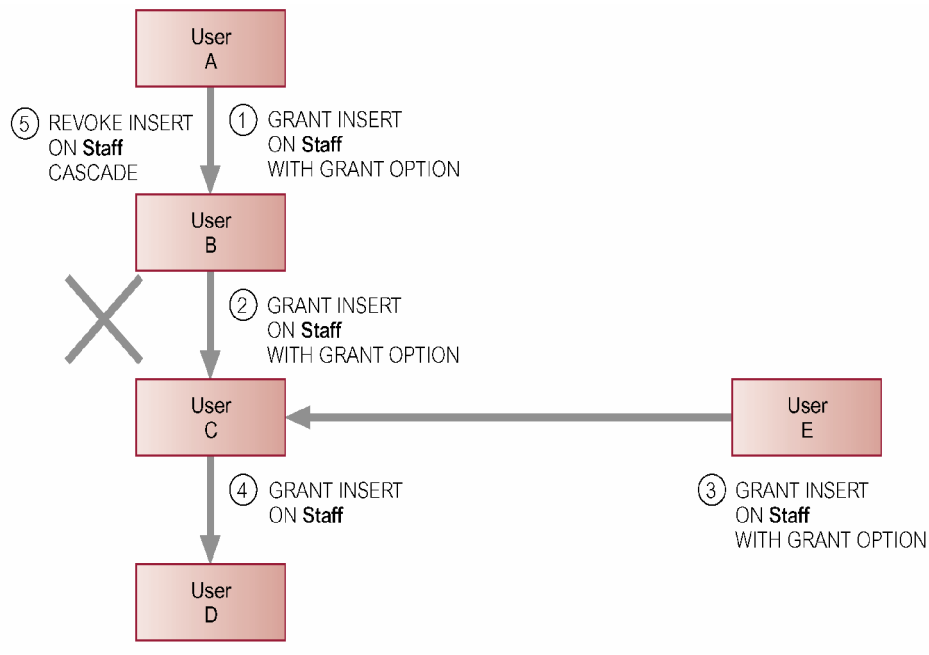
## Data Base Management Systems

```

REVOKE [GRANT OPTION FOR]
      {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
      [RESTRICT | CASCADE]

```

- ◆ ALL PRIVILEGES refers to all privileges granted to a user by user revoking privileges.
- ◆ GRANT OPTION FOR allows privileges passed on via WITH GRANT OPTION of GRANT to be revoked separately from the privileges themselves.
- ◆ REVOKE fails if it results in an abandoned object, such as a view, unless the CASCADE keyword has been specified.
- ◆ Privileges granted to this user by other users are not affected.



### REVOKE Specific Privileges

Revoke privilege SELECT on Branch table from all users.

```

REVOKE SELECT
ON Branch
FROM PUBLIC;

```

Revoke all privileges given to Director on Staff table.

```

REVOKE ALL PRIVILEGES
ON Staff
FROM Director;

```

## Data Base Management Systems

### Chapter 07

#### Overview of Storage and Indexing:

Databases are stored physically as files of records, which are typically stored on magnetic disks.

#### Introduction:

The collection of data that makes up a computerized database must be stored physically on some computer storage medium. The DBMS software that can then retrieve, update, and process this data as needed. Computer storage media form a storage hierarchy that includes two main categories,

- **Primary Storage**

- The category includes storage media that can be operated on directly by the computer Central Processing Unit (CPU), such as the computer main memory and smaller but faster cache memories. Primary storage usually provides fast access to data but is of limited storage capacity.

- **Secondary Storage**

- This category includes magnetic disks, optical disks, and tapes. These devices usually have a larger capacity, cost less, and provide slower access to data than do primary storage devices. Data in secondary storage can not be processed directly by the CPU: it must first be copied into primary storage.

The storage media are classified by the speed with which data can be accessed, by the cost per unit of data to buy the medium, and by the medium's reliability. Let's look into the media that are typically available.

- **Cache:** The cache is the fastest and most costly form of storage. Cache memory is small. The computer hardware manages its use.
- **Main Memory:** The general purpose machine instructions operate on main memory. If a power failure or system crash occurs, the contents of main memory are usually lost. (i.e.) volatile memory type.
- **Flash Memory:** Also known as Electrically Erasable programmable read only memory (EEPROM). Data survive power failure in flash memory (Non-volatile Type). Reading data from flash memory takes less than 100 nanoseconds, which is as fast as reading data from main memory. However writing data to flash memory is more complicated. Data can be written once, but can not be overwritten directly. To overwrite memory that has been written already, we have to erase an entire bank of memory once.

## Data Base Management Systems

It is used for storing small volumes of data (5-10 MB) in hand-held computers, digital cameras.

- **Magnetic Disk Storage:** The primary medium for the long-term on-line storage of data is the magnetic disk. The system must move the data from disk to main memory so that they can be accessed. Disk storage survives power failures and system crashes. Disk-storage devices themselves may sometimes fail and thus destroy data, but such failures do not happen frequently.
- **Optical Storage:** The popular form of optical storage are the compact disk(CD) which can hold 640MB of data, and the digital video disk(DVD), which can hold 4.7 or 8.5 GB of data per side of the disk. Data are stored optically on the disk, and are read by the laser. The optical disks used in read-only compact disks(CD-ROM) or read only digital video disk(DVD-ROM) can not be written, but are supplied with the prerecorded data. There are record one versions of compact disk called (CD-R) and digital video(DVD-R),which can be written once; such records are called write once read many(WORM).There are also multiple write versions of compact disk (CD-RW) and digital video disk(DVD-RW and DVD-RAM), which can be written multiple times.
- **Tape Storage:** It is mainly for back up and archival data. Although magnetic tape is much cheaper than disks, access to data is much slower. Because the tape must be accessed sequentially from the beginning. That's why the tape storage is referred to as sequential-access storage.

### Storage device Hierarchy

Cache

Main Memory

Flash memory

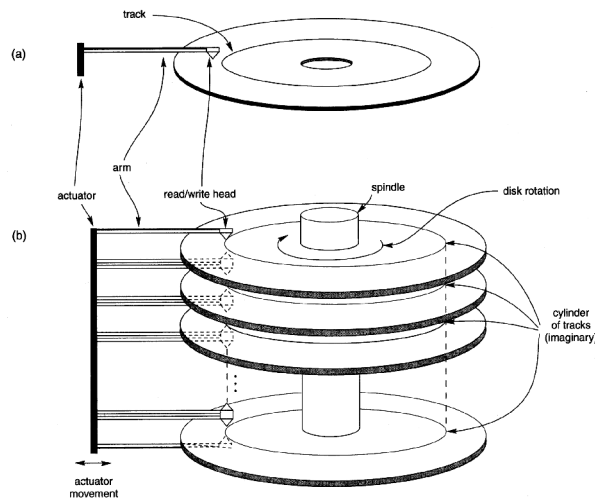
## Data Base Management Systems

Magnetic Disk

Optical Disk

Magnetic Tapes

### Magnetic Disks :



The read write head is positioned very close to the platter surface. We can say almost touching it. It reads or writes magnetically encoded information. The surface of platter is divided into circular tracks. There are over 16000 tracks per platter on typical hard disks. Each track is further divided into sectors. A sector is defined as the smallest unit of data that can be read or written. The size of a sector is typically 512 bytes. There are 200 (on inner disks) to 400 (on outer disks) typical sectors per track. To read or write a sector, the disk arm swings to position head on right track. As the platter spins continually, the data is read or written as sector passes under head. Let us understand the head-disk assemblies as follows: Multiple disk platters are found (typically 2 to 4) on single spindle. There is one head per platter, which is mounted on a common arm. Earlier generation disks were vulnerable to head crashes. Their surface had metal-oxide coatings, which would disintegrate on head crash and damage all data on disk. The current generation disks are less susceptible to such disastrous failures, although individual sectors may get corrupted.

A disk controller interface between the computer system and the disk drive handles high-level commands

## Data Base Management Systems

to read or write sector, and initiates actions, such as moving the disk arm to the right track and actually reading or writing the data. If the data is corrupted, with very high probability stored checksum won't match recomputed checksum. It ensures successful writing by reading back sector after writing it. It performs remapping of bad sectors.

### Placing File Records on Disk:

#### Records:

Data is usually stored in the form of records. Each record consists of a collection of related data values or items, where each value is formed of one or more bytes and corresponds to a particular field of the record.

#### Files, Fixed- Length records and variable length records:

A file is a sequence of records. In many cases all records in a file are of the same record type. If every record in the file has exactly the same size(in bytes), the file is said to be made up of **Fixed file records**. If different records in the file have different sizes, the file is said to be made up of **Variable length records**.

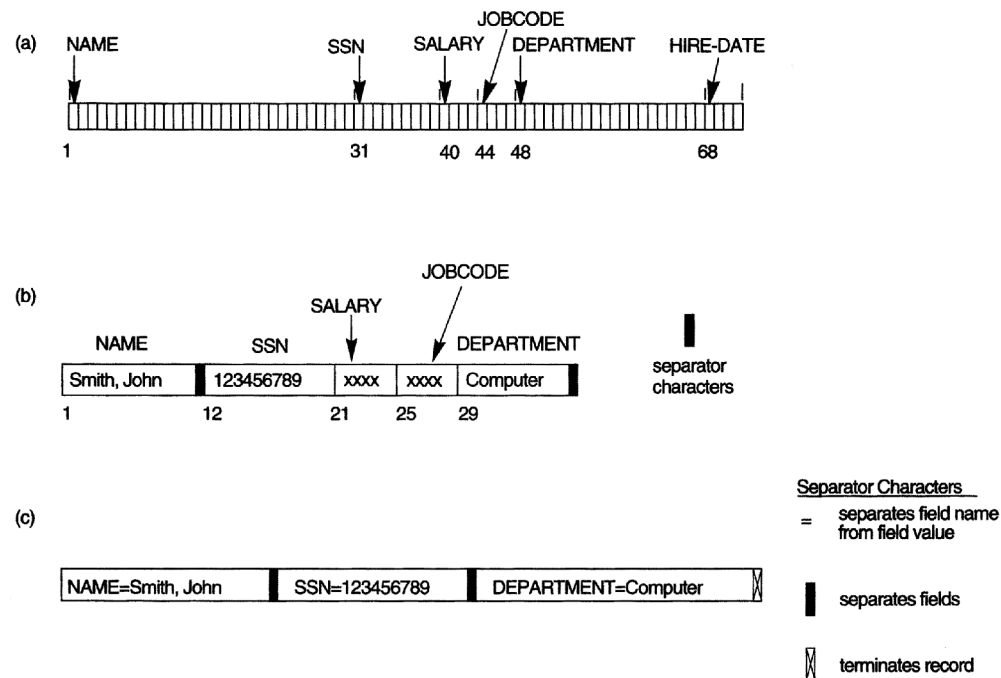


Fig. Three record storage formats.

(a) A fixed-length record with six fields and size of 71 bytes.

(b) A record with two variable-length fields and three fixed-length fields.

(c) A variable-field record with three types of separator

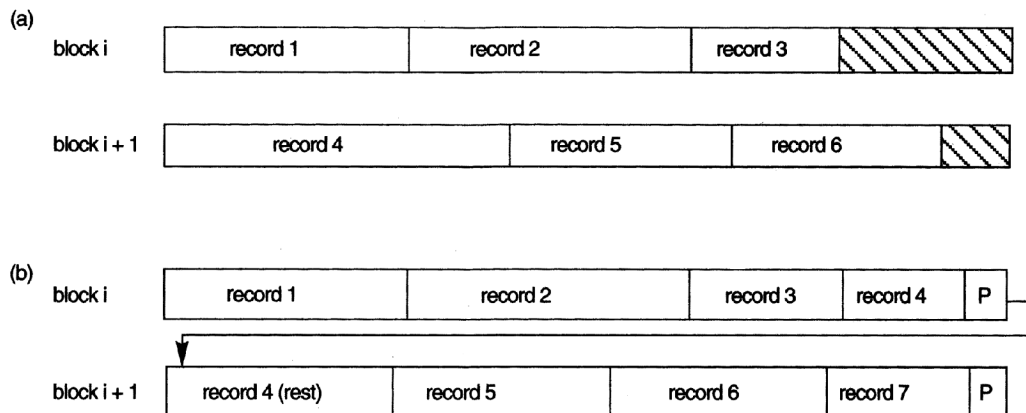


## Data Base Management Systems

### Spanned Vs. Unspanned records :

A pointer at the end of the first block points to the block containing the remainder of the record in case it is not the next consecutive block on disk. This organization is called **Spanned**, because the record can span more than one block

If records are not allowed to cross block boundaries, the organization is called unspanned.



### FIGURE

Types of record organization. (a) Unspanned. (b) Spanned.

### Files of Unordered Records (Heap Files):

In the simplest and most basic type of organization, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organization is called a **heap** or **pile file**. Inserting a new record is very efficient: the last disk block of the file is copied into a buffer, the new record is added; and the block is then rewritten back to the disk. The address of last file block is kept in the file header.

### Files or Ordered Records (Sorted Files)

We can physically order the records of a file on disk based on the value of one of their fields- called the ordering field. This leads to an ordered or sequential file. If the ordering field is also a key field of the file - a field guaranteed to have a unique value in each record- then the field is called the ordering key for the file. Figure shows an ordered file with name as the ordering key field assuming that employees having distinct names.)

## Data Base Management Systems

Some blocks of an ordered (sequential) file of EMPLOYEE records with NAME as the ordering key field

	NAME	SSN	BIRTHDATE	JOB	SALARY	SEX
block 1	Aaron, Ed					
	Abbott, Diane					
	⋮					
	Acosta, Marc					
block 2	Adams, John					
	Adams, Robin					
	⋮					
	Akers, Jan					
block 3	Alexander, Ed					
	Alfred, Bob					
	⋮					
	Allen, Sam					
block 4	Allen, Troy					
	Anders, Keith					
	⋮					
	Anderson, Rob					
block 5	Anderson, Zach					
	Angeli, Joe					
	⋮					
	Archer, Sue					
block 6	Arnold, Mack					
	Arnold, Steven					
	⋮					
	Atkins, Timothy					
	⋮					
block n - 1	Wong, James					
	Wood, Donald					
	⋮					
	Woods, Manny					
block n	Wright, Pam					
	Wyatt, Charles					
	⋮					
	Zimmer, Byron					

### Hashing Technique:

A type of primary file organization is based on hashing, which provides very fast access to records on certain search conditions. This organization is usually called hash file. The search condition is must be an equality condition on a single field, called the hash field of the file. In most cases, the hash field is also a key field of the file, in which case it is called hash Key. The idea behind hashing is to provide a function  $h$ , called a hash function or randomizing function that is applied to the hash field value of a record and yields the address of the disk block in which the record is stored. A search for the record within the block can be carried out in a main memory buffer.

### Internal Hashing:

## Data Base Management Systems

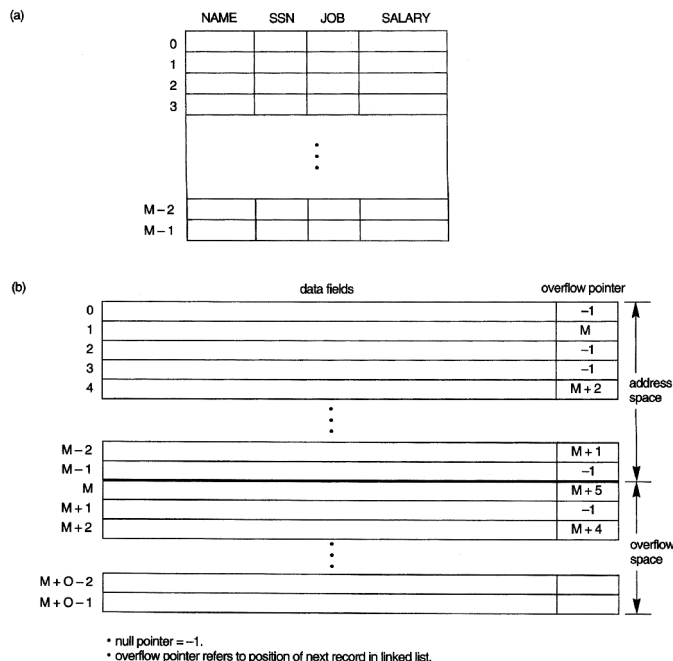
For internal files, hashing is typically implemented as a hash table through the use of an array of records. Suppose that the array index range is from 0 to  $M-1$  then we have  $M$  slots whose address corresponds to the array indexes. We choose a hash function that transforms the hash field value into an integer between 0 and  $M-1$ . One common hash function is the  $h(K)=K \text{ mod } M$  function, which returns the remainder of an integer hash field value  $K$  after division by  $M$ ; this value is then used for the record address.

Non integer hash field values can be transformed into integers before the mod function applied. For characters string, the numeric (ASCII) codes associated with characters can be used in the transformation.

A collision occurs when the hash field value of a record that is being inserted hashes to an address that already contains a different record. In this situation, we must insert the new record in some other position, since its hash address is occupied. The process of finding another position is called **Collision resolution**.

There are numerous methods for collision resolution, including the following,

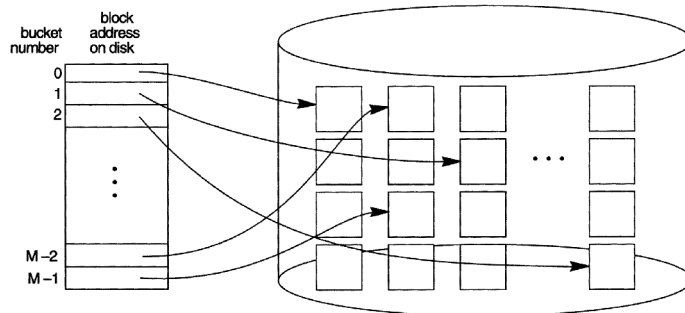
1. Open addressing
2. Chaining
3. Multiple hashing



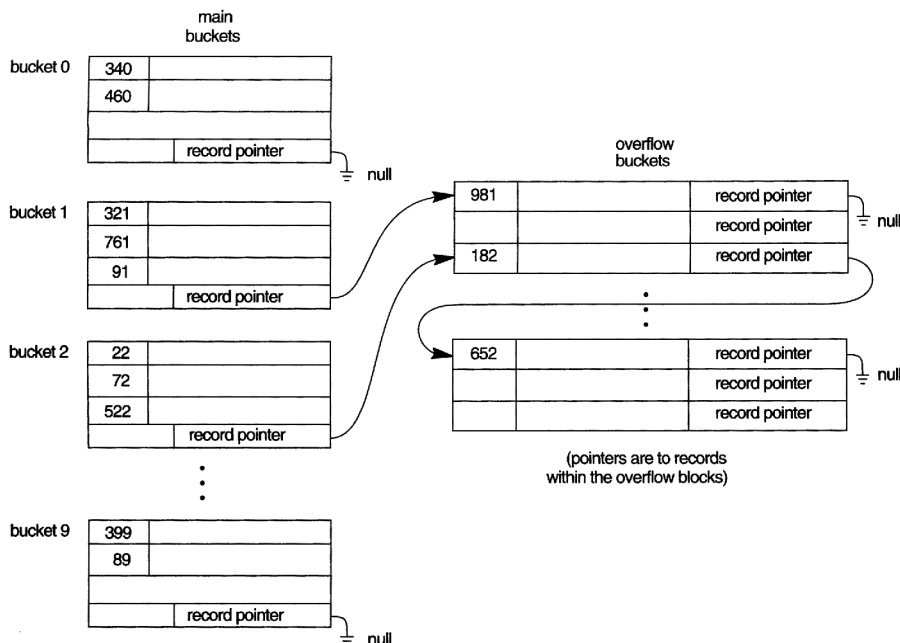
**FIGURE:** Internal hashing data structures.  
(a) Array of  $M$  positions for use in internal hashing.  
(b) Collision resolution by chaining records.

## Data Base Management Systems

Hashing for disk files is called External hashing. To suit the characteristics of disk storage the target address space is made of buckets, each of which hold multiple records. The hashing function maps a key into a relative bucket number, rather than assign an absolute block address to the bucket. A table maintained in the file header converts the bucket number into the corresponding disk block address, as illustrated in figure.



The collision problem is less severe with buckets, because as many records as will fit in bucket can hash to the same bucket without causing problems. However, we must make provisions for the case where a bucket is filled to capacity and a new record being inserted hashes to that bucket. We can use a variation of chaining in which a pointer is maintained in each bucket to a linked list of overflow records for the bucket, as shown in the figure.



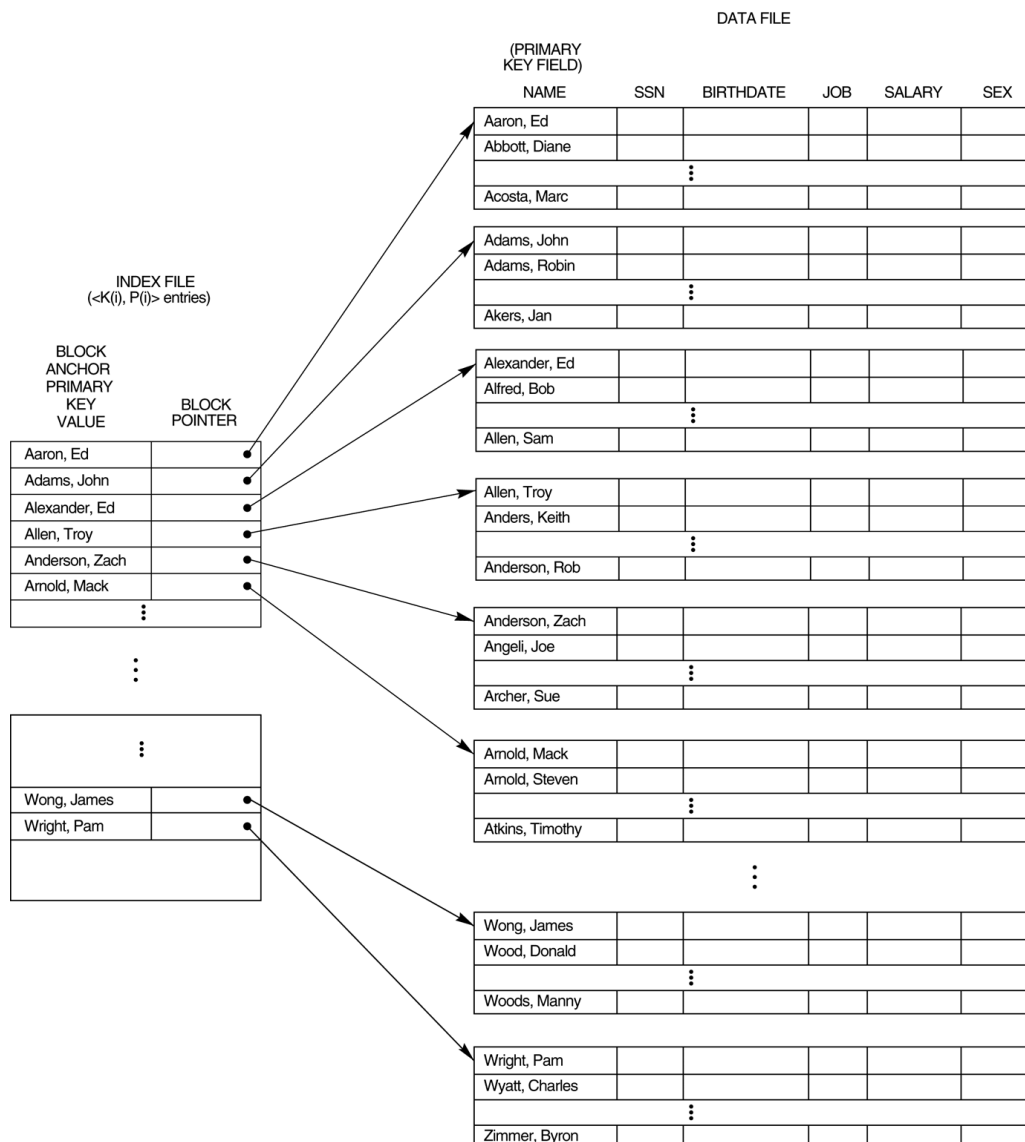
The pointers in the linked list should be record pointers, which include both a block address and a relative record position within the block. [www.jntuworld.com](http://www.jntuworld.com)

## Data Base Management Systems

### Indexing structures for files

#### Primary Indexes (Ordered Indexes)

A primary index is an ordered file whose records are of fixed length with two fields. The first field is of the same data type as the ordering key field- called the primary key- of the data file, and the second field is a pointer to a disk block( a block address). There is one index entry (or index record) in the index file for each block in the data file. Each index entry has the value of the primary key field for the first record in a block and a pointer to that block as its two field values. We will refer to the two field values of index entry  $i$  as  $\langle K(I), P(i) \rangle$



Indexes can also be characterized as dense or sparse. A **dense index** has an index entry for the every search key value( and hence every record) in the data file. A **sparse (or non dense) index** on the other hand, has index entries only for some of the search values. A primary index is hence a non

## Data Base Management Systems

block of the data file and the keys of its anchor record rather than for every search values (or every record).

### Clustering Indexes:

If records of a file are physically ordered on a non key field- which does not have a distinct value for each record- that field is called the clustering field. We can create a different type of index, called clustering index, to speed up the retrieval of records that have the same value for the clustering field. This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record. A clustering index is also an ordered file with two fields.

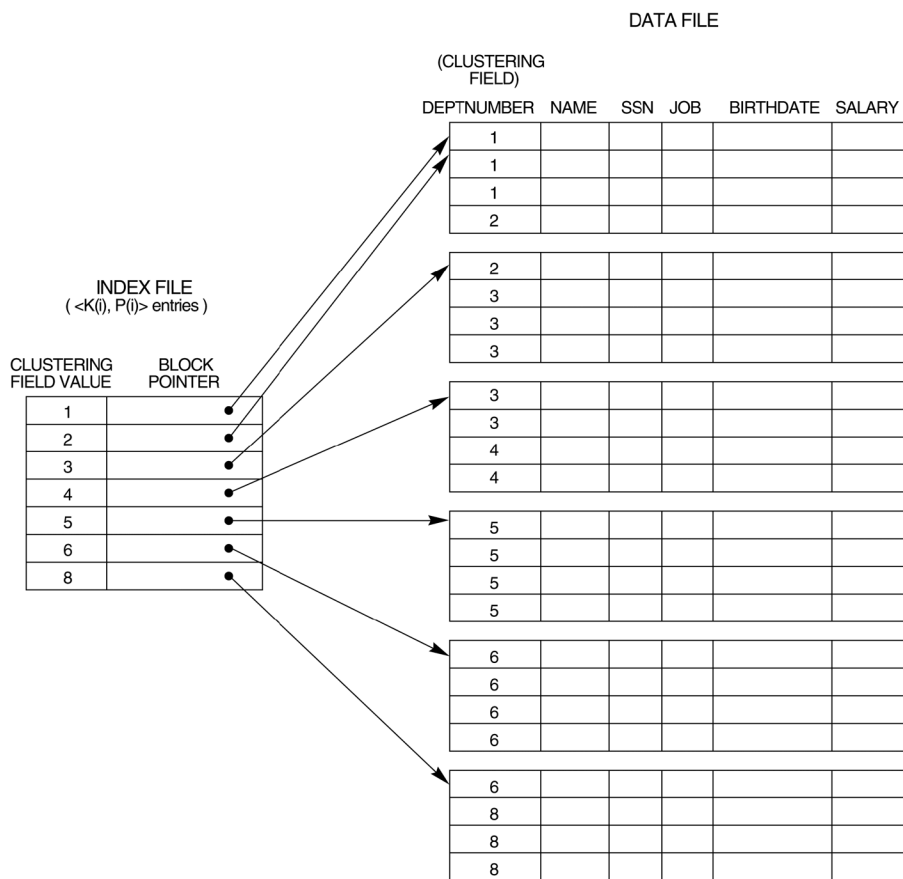


Figure: A clustering index on the DEPTNUMBER ordering non key field of an EMPLOYEE file.

## Data Base Management Systems

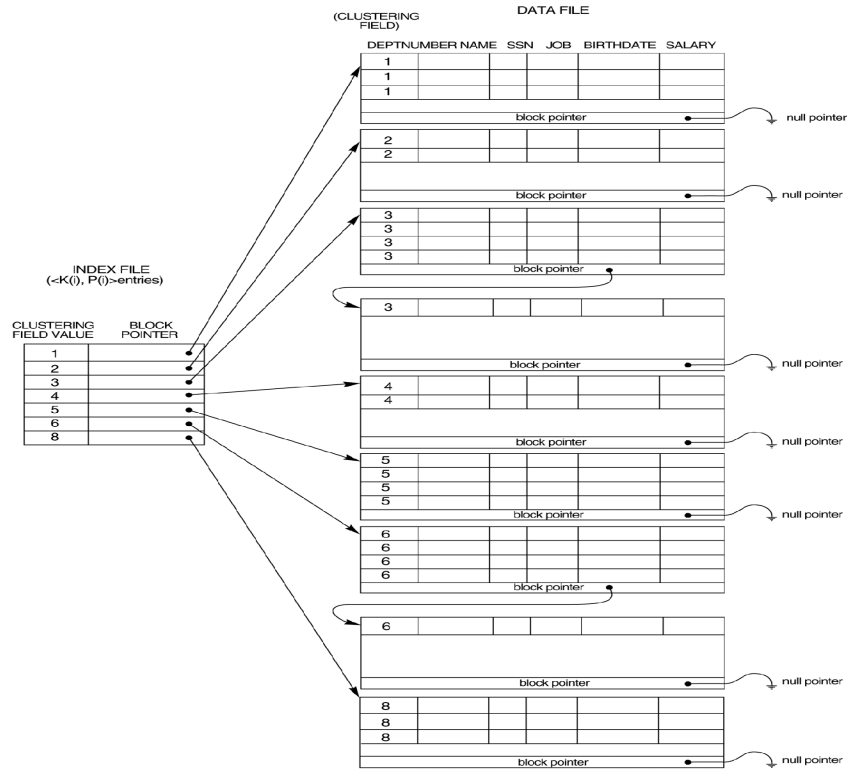


Figure : Clustering index with a separate block cluster for each group of records that share the same value for the clustering field

### Secondary Indexes (Unordered or dense Index)

A secondary index provides a secondary means of accessing a file for which some primary access already exists. The index is an ordered file with two fields. The first field is of the same data type as some non ordering field of the data file that is an indexing field. The second field is either the block pointer or a record pointer. There can be many secondary indexes (and hence indexing field) for the same file.

## Data Base Management Systems

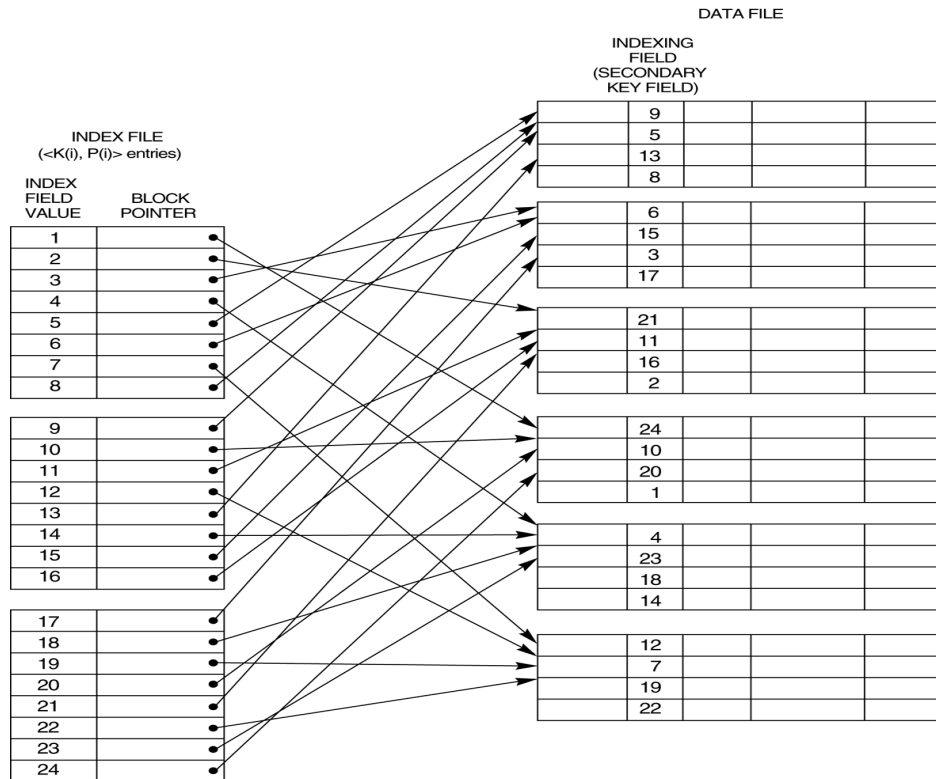


Figure: A dense secondary index (with block pointers) on a nonordering key field of a file.

### Multilevel Indexes:

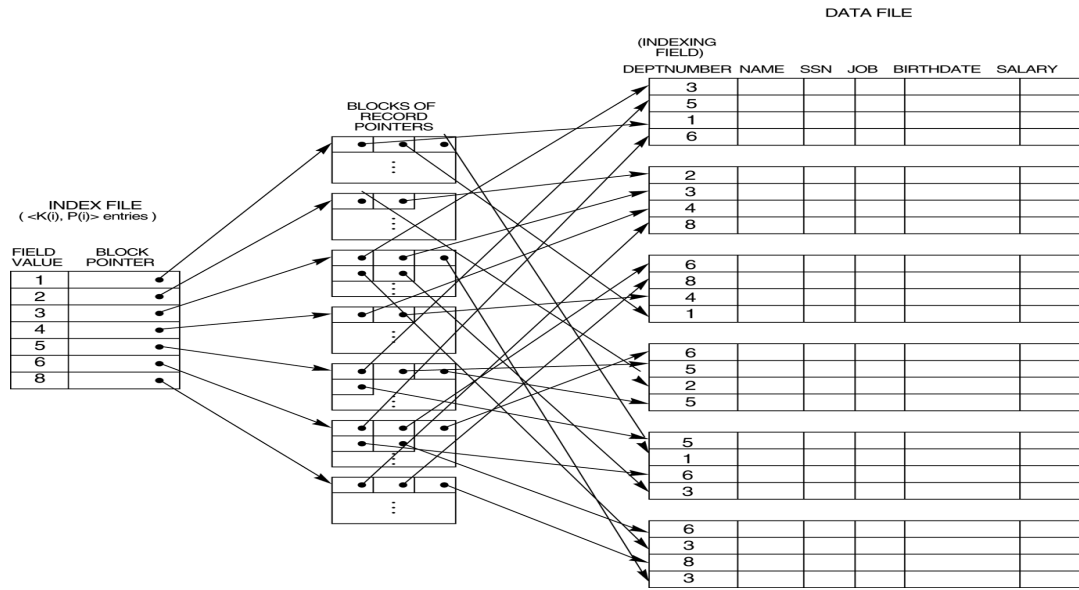
The idea behind the multilevel index is to reduce the part of the index that we continue to search the blocking factor is larger than two. Hence the search space is reduced much faster.

The following figure explains this concept

A secondary index (with recorded pointers) on a non key field implemented using one level of indirection so that index entries are of fixed length and have unique field values.



## Data Base Management Systems



### Chapter 08

#### Query Evaluation overview

##### Introduction

- ◆ In network and hierarchical DBMSs, low-level procedural query language is generally embedded in high-level programming language.
- ◆ Programmer's responsibility to select most appropriate execution strategy.
- ◆ With declarative languages such as SQL, user specifies what data is required rather than how it is to be retrieved.
- ◆ Relieves user of knowing what constitutes good execution strategy.
- ◆ Also gives DBMS more control over system performance.
- ◆ Two main techniques for query optimization:
  - ◆ heuristic rules that order operations in a query;
  - ◆ comparing different strategies based on relative costs, and selecting one that minimizes resource usage.
- ◆ Disk access tends to be dominant cost in query processing for centralized DBMS.

#### Query Processing

Activities involved in retrieving data from the database.

- ◆ Aims of QP:

## Data Base Management Systems

- Transform query written in high-level language (e.g. SQL), into correct and efficient execution strategy expressed in low-level language (implementing RA);
- execute strategy to retrieve required data.

### Query Optimization

Activity of choosing an efficient execution strategy for processing query.

- ◆ As there are many equivalent transformations of same high-level query, aim of QO is to choose one that minimizes resource usage.
- ◆ Generally, reduce total execution time of query.
- ◆ May also reduce response time of query.
- ◆ Problem computationally intractable with large number of relations, so strategy adopted is reduced to finding near optimum solution.

### Example - Different Strategies

Find all Managers who work at a London branch.

```
SELECT *
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
(s.position = 'Manager' AND b.city = 'London');
```

#### ◆ Three equivalent RA queries are:

(1)  $\sigma_{(position='Manager') \wedge (city='London')} \wedge (Staff \bowtie Branch)$

(2)  $\sigma_{(position='Manager') \wedge (city='London')}(Staff \bowtie Branch)$

(3)  $(\sigma_{position='Manager'}(Staff) \bowtie_{Staff.branchNo=Branch.branchNo} (\sigma_{city='London'}(Branch)))$

#### ◆ Assume:

- 1000 tuples in Staff; 50 tuples in Branch;
- 50 Managers; 5 London branches;
- no indexes or sort keys;
- results of any intermediate operations stored on disk;
- cost of the final write is ignored;

## Data Base Management Systems

### Cost Comparison

◆ Cost (in disk accesses) are:

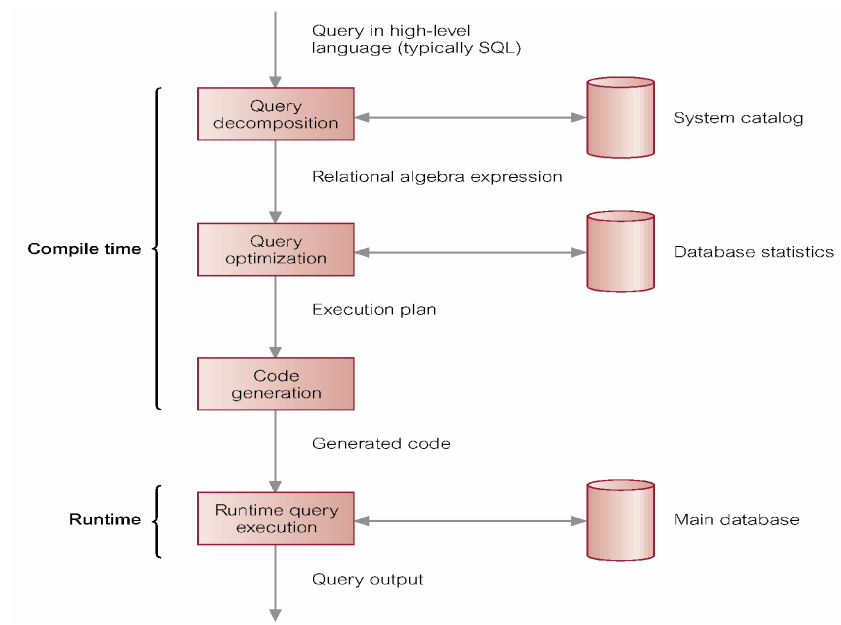
- (1)  $(1000 + 50) + 2*(1000 * 50) = 101\ 050$
- (2)  $2*1000 + (1000 + 50) = 3\ 050$
- (3)  $1000 + 2*50 + 5 + (50 + 5) = 1\ 160$

◆ Cartesian product and join operations much more expensive than selection, and third option significantly reduces size of relations being joined together.

### Phases of Query Processing

◆ QP has four main phases:

- decomposition (consisting of parsing and validation);
- optimization;
- code generation;
- execution.



### Dynamic versus Static Optimization

◆ Two times when first three phases of QP can be carried out:

## Data Base Management Systems

- dynamically every time query is run;
- statically when query is first submitted.
- ◆ Advantages of dynamic QO arise from fact that information is up to date.
- ◆ Disadvantages are that performance of query is affected, time may limit finding optimum strategy.
- ◆ Advantages of static QO are removal of runtime overhead, and more time to find optimum strategy.
- ◆ Disadvantages arise from fact that chosen execution strategy may no longer be optimal when query is run.
- ◆ Could use a hybrid approach to overcome this.

### Cost Estimation for RA Operations

- ◆ Many different ways of implementing RA operations.
- ◆ Aim of QO is to choose most efficient one.
- ◆ Use formulae that estimate costs for a number of options, and select one with lowest cost.
- ◆ Consider only cost of disk access, which is usually dominant cost in QP.
- ◆ Many estimates are based on cardinality of the relation, so need to be able to estimate this.

### Database Statistics

- ◆ Success of estimation depends on amount and currency of statistical information DBMS holds.
- ◆ Keeping statistics current can be problematic.
- ◆ If statistics updated every time tuple is changed, this would impact performance.
- ◆ DBMS could update statistics on a periodic basis, for example nightly, or whenever the system is idle.

### Typical Statistics for Relation R

nTuples(R) - number of tuples in R.

bFactor(R) - blocking factor of R.

nBlocks(R) - number of blocks required to store R:

$nBlocks(R) = \lceil nTuples(R) / bFactor(R) \rceil$

### Query Optimization in Oracle

- ◆ Oracle supports two approaches to query optimization: rule-based and cost-based.

## Data Base Management Systems

### Rule-based

- ◆ 15 rules, ranked in order of efficiency. Particular access path for a table only chosen if statement contains a predicate or other construct that makes that access path available.
- ◆ Score assigned to each execution strategy using these rankings and strategy with best (lowest) score selected.
- ◆ When 2 strategies produce same score, tie-break resolved by making decision based on order in which tables occur in the SQL statement.

**Table 20.4** Rule-based optimization rankings.

Rank	Access path
1	Single row by ROWID (row identifier)
2	Single row by cluster join
3	Single row by hash cluster key with unique or primary key
4	Single row by unique or primary key
5	Cluster join
6	Hash cluster key
7	Indexed cluster key
8	Composite key
9	Single-column indexes
10	Bounded range search on indexed columns
11	Unbounded range search on indexed columns
12	Sort-merge join
13	MAX or MIN of indexed column
14	ORDER BY on indexed columns
15	Full table scan

### OO in Oracle - Rule-based: Example

```
SELECT propertyNo
FROM PropertyForRent
WHERE rooms > 7 AND city = 'London'
```

- ◆ Single-column access path using index on city from WHERE condition (city = 'London'). Rank 9.
- ◆ Unbounded range scan using index on rooms from WHERE condition (rooms > 7). Rank 11.
- ◆ Full table scan - rank 15.
- ◆ Although there is index on propertyNo, column does not appear in WHERE clause and so is not considered by optimizer.
- ◆ Based on these paths, rule-based optimizer will choose to use index based on city column.

## Data Base Management Systems

### QO in Oracle - Cost-Based

- ◆ To improve QO, Oracle introduced cost-based optimizer in Oracle 7, which selects strategy that requires minimal resource use necessary to process all rows accessed by query (avoiding above tie-break anomaly).
- ◆ User can select whether minimal resource usage is based on *throughput* or based on *response time*, by setting the OPTIMIZER\_MODE initialization parameter.
- ◆ Cost-based optimizer also takes into consideration hints that the user may provide.

## Data Base Management Systems

### Chapter 9

#### Transaction Processing:

##### Transaction:

Action, or series of actions, carried out by user or application, which accesses or changes contents of database. It Transforms database from one consistent state to another, although consistency may be violated during transaction.

##### Single User Vs. Multi User Systems:

A DBMS is a single user if at most one user at a time can use the system, and it is multi user if many users can use the system and hence access the database concurrently. Multiple users can access databases and use the computer systems Simultaneously because of the concept of Multiprogramming, which allows the computer to execute multiple programs or processes at the same time. If only a single central processing unit(CPU) exists, it can actually executes at most one process at a time. However multiprogramming operating systems executes some commands from one process then suspend that process and execute some commands from the next process, and so on. A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again. Hence concurrent execution of process is actually interleaved as illustrated in the following figure, which shows two processes A and B executing concurrently in an interleaved fashion.

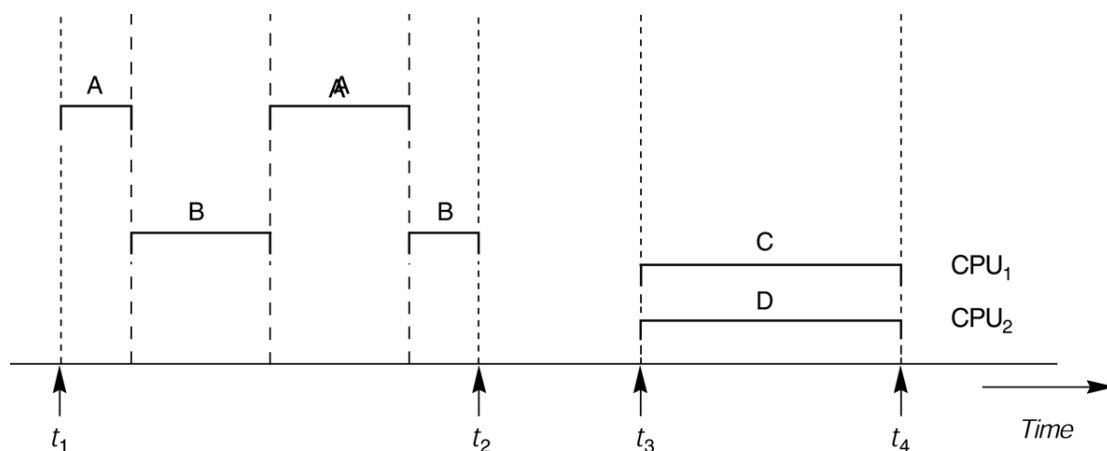


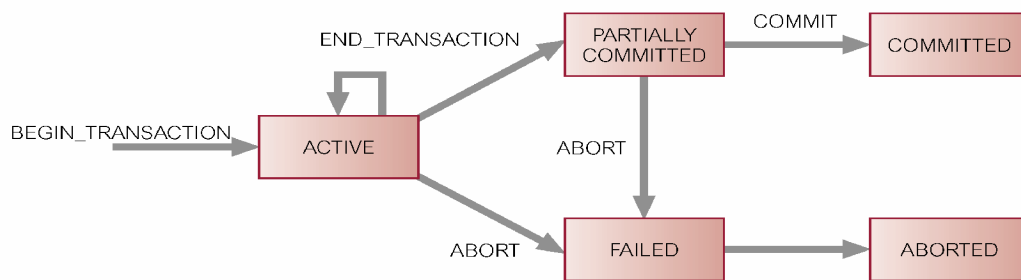
Fig. Interleaved processing Vs. Parallel Processing of concurrent transactions.

Interleaving also prevents the long process from delaying other processes. If the computer system has multiple hardware processors(CPUs), parallel processing of multiple processing is possible as illustrated the process C and D in the figure.

## Data Base Management Systems

- ◆ Can have one of two outcomes:
  - Success - transaction *commits* and database reaches a new consistent state.
  - Failure - transaction *aborts*, and database must be restored to consistent state before it started.
  - Such a transaction is *rolled back* or *undone*.
- ◆ Committed transaction cannot be aborted.
- ◆ Aborted transaction that is rolled back can be restarted later.

### State Transition Diagram for Transaction



### Properties of Transactions

Four basic (*ACID*) properties of a transaction are:

Atomicity :A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

Consistency :Must transform database from one consistent state to another.

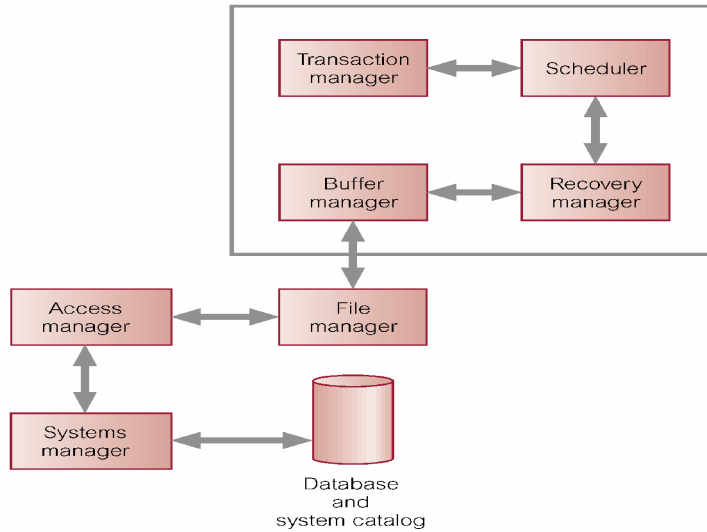
Isolation :Partial effects of incomplete transactions should not be visible to other transactions.

Durability :Effects of a committed transaction are permanent and must not be lost because of later failure.

### DBMS Transaction Subsystem



## Data Base Management Systems



### Concurrency control:

Processes of managing simultaneous operations on the database without having them interfere with one another.

- ◆ Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.
- ◆ Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.

### Need for Concurrency Control

Three examples of potential problems caused by concurrency:

- Lost update problem.
- The temporary update or Dirty Read Problem.
- Incorrect summary problem.

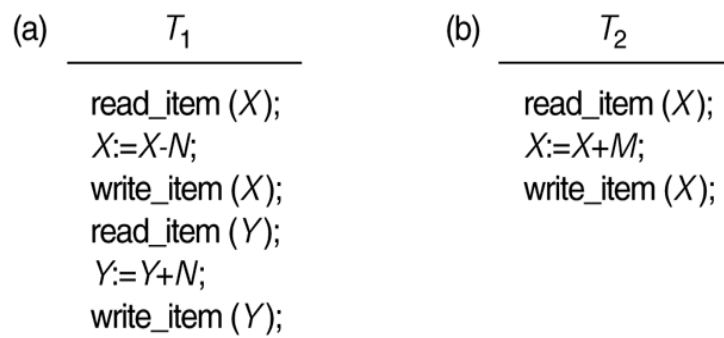


Fig. 1 : Two sample transaction Transaction 1 and Transaction 2

## Data Base Management Systems

The problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of the some database items incorrect. Suppose that transactions  $T_1$  and  $T_2$  are submitted at approximately the same time, and suppose that their operations are interleaved as shown in the figure a, then the final value of  $X$  is incorrect. Because  $T_2$  reads the value of  $X$  before  $T_1$  changes it in the database and hence the updated value resulting from  $T_1$  is lost. For example if  $X=80$  at the start (originally there were 80 reservation on the flight)  $N=5$  ( $T_1$  transfers 5 seats reservations from the flight corresponding to  $X$  to the flight corresponding to  $Y$ ) and  $M=4$  ( $T_2$  reserves 4 seats on  $X$ ), the final result should be  $X=79$ , but the interleaving of operation shown in the figure a it is  $X=84$  because the update in  $T_1$  that removed the 5 seats from  $X$  was lost.

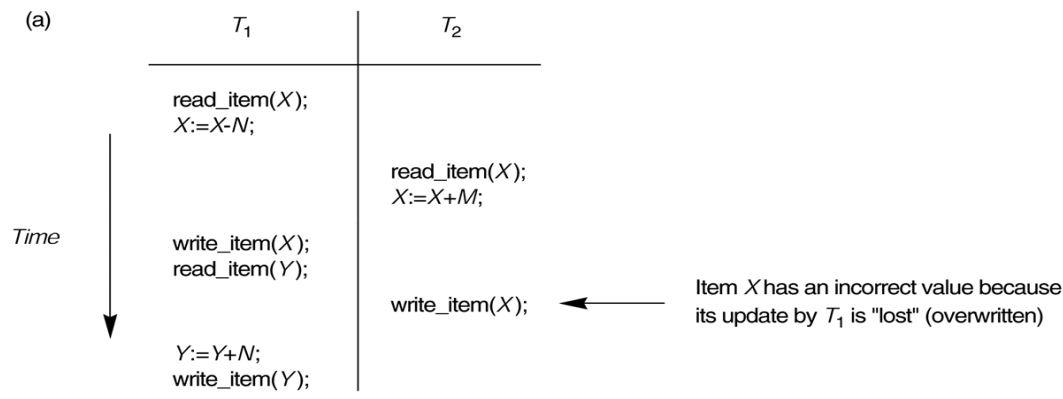


Fig.a. The lost update problem.

### The temporary update or Dirty Read Problem.

This problems occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.

## Data Base Management Systems

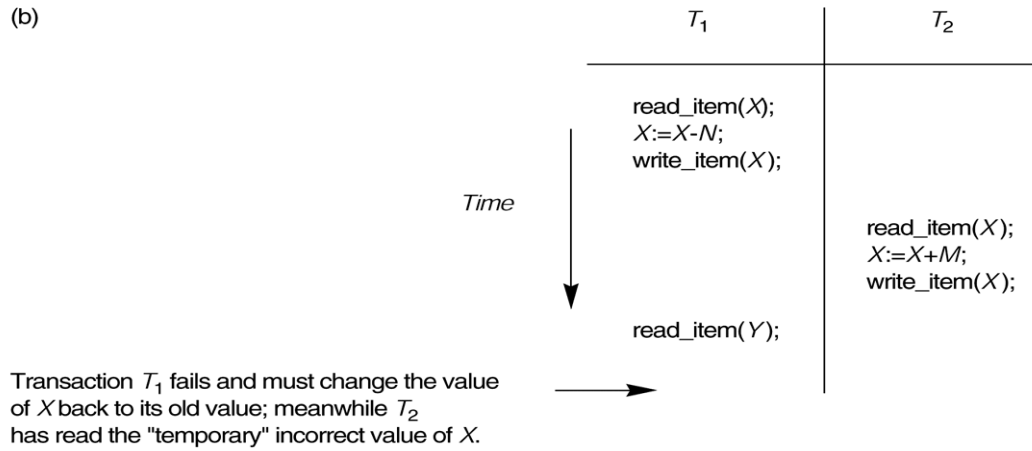
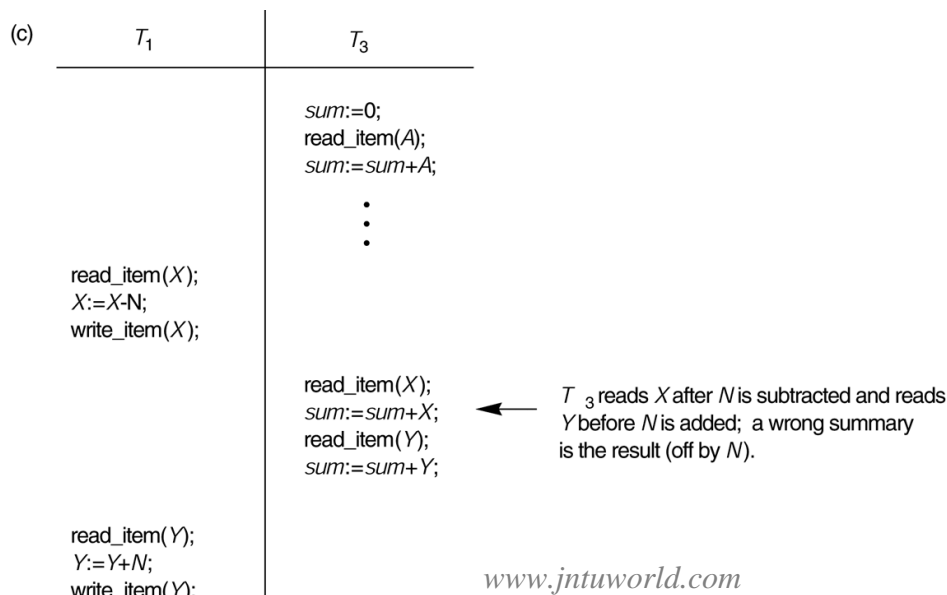


Figure shows an example where  $T_1$  updates items  $X$  then fails before completion, so the system must change  $X$  back to its original. Before it can do so, however transaction  $T_2$  reads the temporary value of  $X$ , which will not be recorded permanently in the database because of the failure of  $T_1$ . The value of item  $X$  that is read by  $T_2$  is called the dirty data.

### Incorrect summary problem

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated. (refer the Elmasri Navathe's Fundamentals of Database Systems for more input)



## Data Base Management Systems

### FIGURE

Some problems that occur when concurrent execution is uncontrolled. (c) The incorrect summary problem.

### Serializability

- ◆ Objective of a concurrency control protocol is to schedule transactions in such a way as to avoid any interference.
- ◆ Could run transactions serially, but this limits degree of concurrency or parallelism in system.
- ◆ Serializability identifies those executions of transactions guaranteed to ensure consistency

### Schedule

Sequence of reads/writes by set of concurrent transactions.

### Serial Schedule

Schedule where operations of each transaction are executed consecutively without any interleaved operations from other transactions.

- ◆ No guarantee that results of all serial executions of a given set of transactions will be identical.

### Nonserial Schedule

- ◆ Schedule where operations from set of concurrent transactions are interleaved.
- ◆ Objective of serializability is to find nonserial schedules that allow transactions to execute concurrently without interfering with one another.
- ◆ In other words, want to find nonserial schedules that are equivalent to *some* serial schedule. Such a schedule is called *serializable*.

### Serializability

- ◆ In serializability, ordering of read/writes is important:
  - (a) If two transactions only read a data item, they do not conflict and order is not important.
  - (b) If two transactions either read or write completely separate data items, they do not conflict and order is not

## Data Base Management Systems

(c) If one transaction writes a data item and another reads or writes same data item, order of execution is important.

### Example of Conflict Serializability

Time	T <sub>7</sub>	T <sub>8</sub>	T <sub>7</sub>	T <sub>8</sub>	T <sub>7</sub>	T <sub>8</sub>
t <sub>1</sub>	begin_transaction		begin_transaction		begin_transaction	
t <sub>2</sub>	read(bal <sub>x</sub> )		read(bal <sub>x</sub> )		read(bal <sub>x</sub> )	
t <sub>3</sub>	write(bal <sub>x</sub> )		write(bal <sub>x</sub> )		write(bal <sub>x</sub> )	
t <sub>4</sub>		begin_transaction		begin_transaction	read(bal <sub>y</sub> )	
t <sub>5</sub>		read(bal <sub>x</sub> )		read(bal <sub>x</sub> )	write(bal <sub>y</sub> )	
t <sub>6</sub>		write(bal <sub>x</sub> )	read(bal <sub>y</sub> )		commit	
t <sub>7</sub>	read(bal <sub>y</sub> )			write(bal <sub>x</sub> )		begin_transaction
t <sub>8</sub>	write(bal <sub>y</sub> )		write(bal <sub>y</sub> )			read(bal <sub>x</sub> )
t <sub>9</sub>	commit		commit			write(bal <sub>x</sub> )
t <sub>10</sub>		read(bal <sub>y</sub> )		read(bal <sub>y</sub> )		read(bal <sub>y</sub> )
t <sub>11</sub>		write(bal <sub>y</sub> )		write(bal <sub>y</sub> )		write(bal <sub>y</sub> )
t <sub>12</sub>		commit		commit		commit
	(a)		(b)		(c)	

### Serializability

- ◆ Conflict serializable schedule orders any conflicting operations in same way as some serial execution.
- ◆ Under *constrained write rule* (transaction updates data item based on its old value, which is first read), use *precedence graph* to test for serializability.

### Precedence Graph

- ◆ Create:
  - node for each transaction;
  - a directed edge  $T_i \rightarrow T_j$ , if  $T_j$  reads the value of an item written by  $T_i$ ;
  - a directed edge  $T_i \rightarrow T_j$ , if  $T_j$  writes a value into an item after it has been read by  $T_i$ .
- ◆ If precedence graph contains cycle schedule is not conflict serializable.

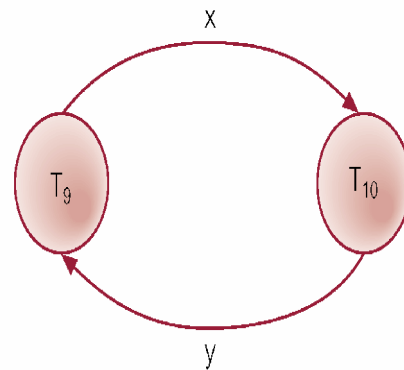
### Example - Non-conflict serializable schedule

- ◆ T<sub>9</sub> is transferring £100 from one account with balance bal<sub>x</sub> to another account with balance bal<sub>y</sub>.
- ◆ T<sub>10</sub> is increasing balance of these two accounts by 10%.
- ◆ Precedence graph has a cycle and so is not serializable.

### Example - Non-conflict serializable schedule

## Data Base Management Systems

Time	T <sub>9</sub>	T <sub>10</sub>
t <sub>1</sub>	begin_transaction	
t <sub>2</sub>	read(bal <sub>x</sub> )	
t <sub>3</sub>	bal <sub>x</sub> = bal <sub>x</sub> + 100	
t <sub>4</sub>	write(bal <sub>x</sub> )	
t <sub>5</sub>		begin_transaction
t <sub>6</sub>		read(bal <sub>x</sub> )
t <sub>7</sub>		bal <sub>x</sub> = bal <sub>x</sub> * 1.1
t <sub>8</sub>		write(bal <sub>x</sub> )
t <sub>9</sub>		read(bal <sub>y</sub> )
t <sub>10</sub>		bal <sub>y</sub> = bal <sub>y</sub> * 1.1
t <sub>11</sub>	read(bal <sub>y</sub> )	
t <sub>12</sub>	bal <sub>y</sub> = bal <sub>y</sub> - 100	
t <sub>13</sub>	write(bal <sub>y</sub> )	
t <sub>14</sub>	commit	



### View Serializability

- ◆ Offers less stringent definition of schedule equivalence than conflict serializability.
- ◆ Two schedules S1 and S2 are view equivalent if:
  - For each data item  $x$ , if  $T_i$  reads initial value of  $x$  in S1,  $T_i$  must also read initial value of  $x$  in S2.
  - For each read on  $x$  by  $T_i$  in S1, if value read by  $x$  is written by  $T_j$ ,  $T_i$  must also read value of  $x$  produced by  $T_j$  in S2.
  - For each data item  $x$ , if last write on  $x$  performed by  $T_i$  in S1, same transaction must perform final write on  $x$  in S2.
- ◆ Schedule is view serializable if it is view equivalent to a serial schedule.
- ◆ Every conflict serializable schedule is view serializable, although converse is not true.
- ◆ It can be shown that any view serializable schedule that is not conflict serializable contains one or more blind writes.
- ◆ In general, testing whether schedule is serializable is NP-complete.

### Example - View Serializable schedule

## Data Base Management Systems

Time	T <sub>11</sub>	T <sub>12</sub>	T <sub>13</sub>
t <sub>1</sub>	begin_transaction		
t <sub>2</sub>	read(bal <sub>x</sub> )		
t <sub>3</sub>		begin_transaction	
t <sub>4</sub>		write(bal <sub>x</sub> )	
t <sub>5</sub>		commit	
t <sub>6</sub>	write(bal <sub>x</sub> )		
t <sub>7</sub>	commit		
t <sub>8</sub>			begin_transaction
t <sub>9</sub>			write(bal <sub>x</sub> )
t <sub>10</sub>			commit

### Recoverability

- ◆ Serializability identifies schedules that maintain database consistency, assuming no transaction fails.
- ◆ Could also examine recoverability of transactions within schedule.
- ◆ If transaction fails, atomicity requires effects of transaction to be undone.
- ◆ Durability states that once transaction commits, its changes cannot be undone (without running another, compensating, transaction).

### Concurrency Control Techniques

- ◆ Two basic concurrency control techniques:
  - Locking,
  - Timestamping.
- ◆ Both are conservative approaches: delay transactions in case they conflict with other transactions.
- ◆ Optimistic methods assume conflict is rare and only check for conflicts at commit.

### Locking

Transaction uses locks to deny access to other transactions and so prevent incorrect updates.

- ◆ Most widely used approach to ensure serializability.
- ◆ Generally, a transaction must claim a *shared* (read) or *exclusive* (write) lock on a data item before read or write.
- ◆ Lock prevents another transaction from modifying item or even reading it, in the case of a write lock.

### Locking - Basic Rules

- ◆ If transaction has *shared* lock on item, can read but not

## Data Base Management Systems

- ◆ If transaction has exclusive lock on item, can both read and update item.
- ◆ Reads cannot conflict, so more than one transaction can hold shared locks simultaneously on same item.
- ◆ Exclusive lock gives transaction exclusive access to that item.
- ◆ Some systems allow transaction to upgrade read lock to an exclusive lock, or downgrade exclusive lock to a shared lock.

### Example - Incorrect Locking Schedule

- ◆ For two transactions above, a valid schedule using these rules is:

```
S = {write_lock(T9, balx), read(T9, balx), write(T9, balx),
unlock(T9, balx), write_lock(T10, balx), read(T10, balx),
write(T10, balx), unlock(T10, balx), write_lock(T10, baly),
read(T10, baly), write(T10, baly), unlock(T10, baly),
commit(T10), write_lock(T9, baly), read(T9, baly), write(T9,
baly), unlock(T9, baly), commit(T9) }
```

### Example - Incorrect Locking Schedule

- ◆ If at start, balx = 100, baly = 400, result should be:
  - balx = 220, baly = 330, if T9 executes before T10, or
  - balx = 210, baly = 340, if T10 executes before T9.
- ◆ However, result gives balx = 220 and baly = 340.
- ◆ S is not a serializable schedule.
- ◆ Problem is that transactions release locks too soon, resulting in loss of total isolation and atomicity.
- ◆ To guarantee serializability, need an additional protocol concerning the positioning of lock and unlock operations in every transaction.

### Two-Phase Locking (2PL)

Transaction follows 2PL protocol if all locking operations precede first unlock operation in the transaction.

- ◆ Two phases for transaction:
  - Growing phase - acquires all locks but cannot release any locks.
  - Shrinking phase - releases locks but cannot acquire any new locks



## Data Base Management Systems

### Preventing Lost Update Problem using 2PL

Time	T <sub>1</sub>	T <sub>2</sub>	bal <sub>x</sub>
t <sub>1</sub>		begin_transaction	100
t <sub>2</sub>	begin_transaction	write_lock(bal <sub>x</sub> )	100
t <sub>3</sub>	write_lock(bal <sub>x</sub> )	read(bal <sub>x</sub> )	100
t <sub>4</sub>	WAIT	bal <sub>x</sub> = bal <sub>x</sub> + 100	100
t <sub>5</sub>	WAIT	write(bal <sub>x</sub> )	200
t <sub>6</sub>	WAIT	commit/unlock(bal <sub>x</sub> )	200
t <sub>7</sub>	read(bal <sub>x</sub> )		200
t <sub>8</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10		200
t <sub>9</sub>	write(bal <sub>x</sub> )		190
t <sub>10</sub>	commit/unlock(bal <sub>x</sub> )		190

### Deadlock

An impasse that may result when two (or more) transactions are each waiting for locks held by the other to be released.

Time	T <sub>17</sub>	T <sub>18</sub>
t <sub>1</sub>	begin_transaction	
t <sub>2</sub>	write_lock(bal <sub>x</sub> )	begin_transaction
t <sub>3</sub>	read(bal <sub>x</sub> )	write_lock(bal <sub>y</sub> )
t <sub>4</sub>	bal <sub>x</sub> = bal <sub>x</sub> - 10	read(bal <sub>y</sub> )
t <sub>5</sub>	write(bal <sub>x</sub> )	bal <sub>y</sub> = bal <sub>y</sub> + 100
t <sub>6</sub>	write_lock(bal <sub>y</sub> )	write(bal <sub>y</sub> )
t <sub>7</sub>	WAIT	write_lock(bal <sub>x</sub> )
t <sub>8</sub>	WAIT	WAIT
t <sub>9</sub>	WAIT	WAIT
t <sub>10</sub>	:	WAIT
t <sub>11</sub>	:	:

- ◆ Only one way to break deadlock: abort one or more of the transactions.
- ◆ Deadlock should be transparent to user, so DBMS should restart transaction(s).
- ◆ Three general techniques for handling deadlock:
  - Timeouts.
  - Deadlock prevention.
  - Deadlock detection and recovery.

### Timeouts

- ◆ Transaction that requests lock will only wait for a system-defined period of time.
- ◆ If lock has not been granted within this period, lock request times out.
- ◆ In this case, DBMS assumes transaction may be deadlocked, even though it may not be, and it aborts and automatically restarts the transaction.

## Data Base Management Systems

### Deadlock Prevention

- ◆ DBMS looks ahead to see if transaction would cause deadlock and never allows deadlock to occur.
- ◆ Could order transactions using transaction timestamps:
  - Wait-Die - only an older transaction can wait for younger one, otherwise transaction is aborted (*dies*) and restarted with same timestamp.
  - Wound-Wait - only a younger transaction can wait for an older one. If older transaction requests lock held by younger one, younger one is aborted (*wounded*).

### Recovery from Deadlock Detection

- ◆ Several issues:
  - choice of deadlock victim;
  - how far to roll a transaction back;
  - avoiding starvation.

### Timestamping

- ◆ Transactions ordered globally so that older transactions, transactions with *smaller* timestamps, get priority in the event of conflict.
- ◆ Conflict is resolved by rolling back and restarting transaction.
- ◆ No locks so no deadlock.

### Timestamp

A unique identifier created by DBMS that indicates relative starting time of a transaction.

- ◆ Can be generated by using system clock at time transaction started, or by incrementing a logical counter every time a new transaction starts.
- ◆ Read/write proceeds only if *last update on that data item* was carried out by an older transaction.
- ◆ Otherwise, transaction requesting read/write is restarted and given a new timestamp.
- ◆ Also timestamps for data items:
  - read-timestamp - timestamp of last transaction to read item;
  - write-timestamp - timestamp **of last transaction to write item.**

### Optimistic Techniques

## Data Base Management Systems

- ◆ Based on assumption that conflict is rare and more efficient to let transactions proceed without delays to ensure serializability.
- ◆ At commit, check is made to determine whether conflict has occurred.
- ◆ If there is a conflict, transaction must be rolled back and restarted.
- ◆ Potentially allows greater concurrency than traditional protocols.
  
- ◆ Three phases:
  - Read
  - Validation
  - Write

### Optimistic Techniques - Read Phase

- ◆ Extends from start until immediately before commit.
- ◆ Transaction reads values from database and stores them in local variables. Updates are applied to a local copy of the data.

### Optimistic Techniques - Validation Phase

- ◆ Follows the read phase.
- ◆ For read-only transaction, checks that data read are still current values. If no interference, transaction is committed, else aborted and restarted.
- ◆ For update transaction, checks transaction leaves database in a consistent state, with serializability maintained.

### Optimistic Techniques - Write Phase

- ◆ Follows successful validation phase for update transactions.
- ◆ Updates made to local copy are applied to the database.

### Granularity of Data Items

- ◆ Size of data items chosen as unit of protection by concurrency control protocol.
- ◆ Ranging from coarse to fine:
  - The entire database.

## Data Base Management Systems

- A page (or area or database spaced).
- A record.
- A field value of a record.

### Granularity of Data Items

- ◆ Tradeoff:
  - coarser, the lower the degree of concurrency;
  - finer, more locking information that is needed to be stored.
- ◆ Best item size depends on the types of transactions

### Hierarchy of Granularity

- ◆ Could represent granularity of locks in a hierarchical structure.
- ◆ Root node represents entire database, level 1s represent files, etc.
- ◆ When node is locked, all its descendants are also locked.
- ◆ DBMS should check hierarchical path before granting lock.
- ◆ *Intention lock* could be used to lock all ancestors of a locked node.
- ◆ Intention locks can be read or write. Applied top-down, released bottom-up.

### Database Recovery

Process of restoring database to a correct state in the event of a failure.

- ◆ Need for Recovery Control
  - Two types of storage: volatile (main memory) and nonvolatile.
  - Volatile storage does not survive system crashes.
  - Stable storage represents information that has been replicated in several nonvolatile storage media with independent failure modes.

### Types of Failures

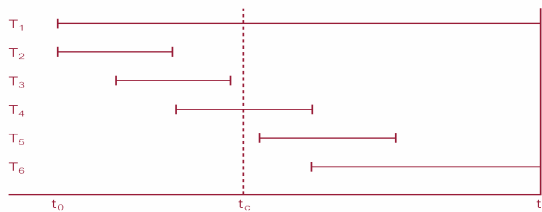
- ◆ System crashes, resulting in loss of main memory.
- ◆ Media failures, resulting in loss of parts of secondary storage.
- ◆ Application software errors.
- ◆ Natural physical disasters.
- ◆ Carelessness or unintentional destruction of data or facilities.

## Data Base Management Systems

### Transactions and Recovery

- ◆ Transactions represent basic unit of recovery.
- ◆ Recovery manager responsible for atomicity and durability.
- ◆ If failure occurs between commit and database buffers being flushed to secondary storage then, to ensure durability, recovery manager has to redo (rollforward) transaction's updates.
- ◆ If transaction had not committed at failure time, recovery manager has to undo (rollback) any effects of that transaction for atomicity.
- ◆ Partial undo - only one transaction has to be undone.
- ◆ Global undo - all transactions have to be undone.

### Example



- ◆ DBMS starts at time  $t_0$ , but fails at time  $t_f$ . Assume data for transactions T2 and T3 have been written to secondary storage.
- ◆ T1 and T6 have to be undone. In absence of any other information, recovery manager has to redo T2, T3, T4, and T5.

### Recovery Facilities

- ◆ DBMS should provide following facilities to assist with recovery:
  - Backup mechanism, which makes periodic backup copies of database.
  - Logging facilities, which keep track of current state of transactions and database changes.
  - Checkpoint facility, which enables updates to database in progress to be made permanent.
  - Recovery manager, which allows DBMS to restore database to consistent state following a failure.

### Log File

- ◆ Contains information about all updates to database:
  - Transaction records.
  - Checkpoint records.
- ◆ Often used for other purposes (for example auditing)

## Data Base Management Systems

- ◆ Transaction records contain:
  - Transaction identifier.
  - Type of log record, (transaction start, insert, update, delete, abort, commit).
  - Identifier of data item affected by database action (insert, delete, and update operations).
  - Before-image of data item.
  - After-image of data item.
  - Log management information.

### Sample Log File

Tid	Time	Operation	Object	Before image	After image	pPtr	nPtr
T1	10:12	START				0	2
T1	10:13	UPDATE	STAFF SL21	(old value)	(new value)	1	8
T2	10:14	START				0	4
T2	10:16	INSERT	STAFF SG37		(new value)	3	5
T2	10:17	DELETE	STAFF SA9	(old value)		4	6
T2	10:17	UPDATE	PROPERTY PG16	(old value)	(new value)	5	9
T3	10:18	START				0	11
T1	10:18	COMMIT				2	0
	10:19	CHECKPOINT	T2, T3				
T2	10:19	COMMIT				6	0
T3	10:20	INSERT	PROPERTY PG4		(new value)	7	12
T3	10:21	COMMIT				11	0

- ◆ Log file may be duplexed or triplexed.
- ◆ Log file sometimes split into two separate random-access files.
- ◆ Potential bottleneck; critical in determining overall performance.
- ◆

### Checkpoint

Point of synchronization between database and log file. All buffers are force-written to secondary storage.

- ◆ Checkpoint record is created containing identifiers of all active transactions.
- ◆ When failure occurs, redo all transactions that committed since the checkpoint and undo all transactions active at time of crash.
- ◆ In previous example, with checkpoint at time  $t_c$ , changes made by T2 and T3 have been written to secondary storage.
- ◆ Thus:
  - ◆ only redo T4 and T5

## Data Base Management Systems

### Recovery Techniques

- ◆ If database has been damaged:
  - Need to restore last backup copy of database and reapply updates of committed transactions using log file.
- ◆ If database is only inconsistent:
  - Need to undo changes that caused inconsistency. May also need to redo some transactions to ensure updates reach secondary storage.
  - Do not need backup, but can restore database using before- and after-images in the log file.

### Main Recovery Techniques

- ◆ Three main recovery techniques:
  - Deferred Update
  - Immediate Update
  - Shadow Paging

#### Deferred Update

- ◆ Updates are not written to the database until after a transaction has reached its commit point.
- ◆ If transaction fails before commit, it will not have modified database and so no undoing of changes required.
- ◆ May be necessary to redo updates of committed transactions as their effect may not have reached database.

#### Immediate Update

- ◆ Updates are applied to database as they occur.
- ◆ Need to redo updates of committed transactions following a failure.
- ◆ May need to undo effects of transactions that had not committed at time of failure.
- ◆ Essential that log records are written before write to database. *Write-ahead log protocol.*
- ◆ If no "transaction commit" record in log, then that transaction was active at failure and must be undone.
- ◆ Undo operations are performed *in reverse order in which they were written to log.*

#### Shadow Paging

- ◆ Maintain two page tables during life of a transaction: current page and shadow page.

## Data Base Management Systems

- ◆ When transaction starts, two pages are the same.
- ◆ Shadow page table is never changed thereafter and is used to restore database in event of failure.
- ◆ During transaction, current page table records all updates to database.
- ◆ When transaction completes, current page table becomes shadow page table.

### Advanced Transaction Models

- ◆ Protocols considered so far are suitable for types of transactions that arise in traditional business applications, characterized by:
  - Data has many types, each with small number of instances.
  - Designs may be very large.
  - Design is not static but evolves through time.
  - Updates are far-reaching.
  - Cooperative engineering.
- ◆ May result in transactions of long duration, giving rise to following problems:
  - More susceptible to failure - need to minimize amount of work lost.
  - May access large number of data items - concurrency limited if data inaccessible for long periods.
  - Deadlock more likely.
  - Cooperation through use of shared data items restricted by traditional concurrency protocols.

## Chapter 10

### Normalization

- ◆ Main objective in developing a logical data model for relational database systems is to create an accurate representation of the data, its relationships, and constraints.
- ◆ To achieve this objective, must identify a suitable set of relations.
- ◆ Four most commonly used normal forms are first (1NF), second (2NF) and third (3NF) normal forms, and Boyce-Codd normal form (BCNF).



## Data Base Management Systems

- ◆ Based on functional dependencies among the attributes of a relation.
- ◆ A relation can be normalized to a specific form to prevent possible occurrence of update anomalies.

### Data Redundancy

- ◆ Major aim of relational database design is to group attributes into relations to minimize data redundancy and reduce file storage space required by base relations.
- ◆ Problems associated with data redundancy are illustrated by comparing the following Staff and Branch relations with the StaffBranch relation.

### Data Redundancy

Staff

staffNo	sName	position	salary	branchNo
SL21	John White	Manager	30000	B005
SG37	Ann Beech	Assistant	12000	B003
SG14	David Ford	Supervisor	18000	B003
SA9	Mary Howe	Assistant	9000	B007
SG5	Susan Brand	Manager	24000	B003
SL41	Julie Lee	Assistant	9000	B005

Branch

branchNo	bAddress
B005	22 Deer Rd, London
B007	16 Argyll St, Aberdeen
B003	163 Main St, Glasgow

StaffBranch

staffNo	sName	position	salary	branchNo	bAddress
SL21	John White	Manager	30000	B005	22 Deer Rd, London
SG37	Ann Beech	Assistant	12000	B003	163 Main St, Glasgow
SG14	David Ford	Supervisor	18000	B003	163 Main St, Glasgow
SA9	Mary Howe	Assistant	9000	B007	16 Argyll St, Aberdeen
SG5	Susan Brand	Manager	24000	B003	163 Main St, Glasgow
SL41	Julie Lee	Assistant	9000	B005	22 Deer Rd, London

- ◆ StaffBranch relation has redundant data: details of a branch are repeated for every member of staff.
- ◆ In contrast, branch information appears only once for each branch in Branch relation and only branchNo is repeated in Staff relation, to represent where each member of staff works.

### Update Anomalies

## Data Base Management Systems

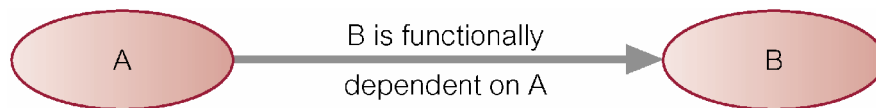
- ◆ Relations that contain redundant information may potentially suffer from update anomalies.
- ◆ Types of update anomalies include:
  - Insertion,
  - Deletion,
  - Modification.

### Lossless-join and Dependency Preservation Properties

- ◆ Two important properties of decomposition:
  - *Lossless-join property* enables us to find any instance of original relation from corresponding instances in the smaller relations.
  - *Dependency preservation property* enables us to enforce a constraint on original relation by enforcing some constraint on each of the smaller relations.

### Functional Dependency

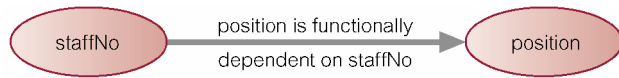
- ◆ Main concept associated with normalization.
- ◆ Functional Dependency
  - Describes relationship between attributes in a relation.
  - If A and B are attributes of relation R, B is functionally dependent on A (denoted  $A \twoheadrightarrow B$ ), if each value of A in R is associated with exactly one value of B in R.
- ◆ Property of the meaning (or semantics) of the attributes in a relation.
- ◆ **Diagrammatic representation:**



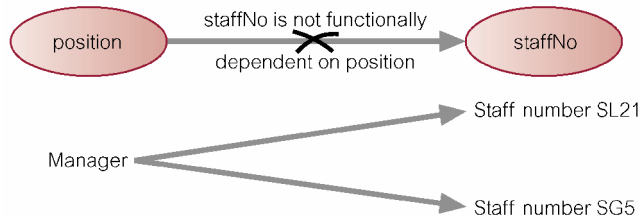
- ◆ *Determinant* of a functional dependency refers to attribute or group of attributes on left-hand side of the arrow.

### Example - Functional Dependency

## Data Base Management Systems



Staff number SL21 → Manager  
(a)



(b)

◆ Main characteristics of functional dependencies used in normalization:

- have a 1:1 relationship between attribute(s) on left and right-hand side of a dependency;
- hold for all time;
- are nontrivial.

◆ Complete set of functional dependencies for a given relation can be very large.

◆ Important to find an approach that can reduce set to a manageable size.

◆ Need to identify set of functional dependencies (X) for a relation that is smaller than complete set of functional dependencies (Y) for that relation and has property that every functional dependency in Y is implied by functional dependencies in X.

◆ Set of all functional dependencies implied by a given set of functional dependencies X called closure of X (written X<sup>+</sup>).

◆ Set of inference rules, called Armstrong's axioms, specifies how new functional dependencies can be inferred from given ones.

◆ Let A, B, and C be subsets of the attributes of relation R. Armstrong's axioms are as follows:

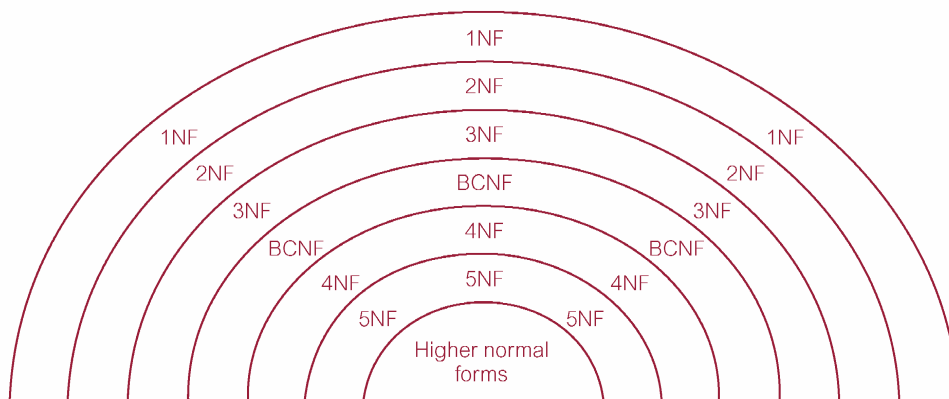
1. Reflexivity If B is a subset of A, then  $A \rightarrow B$
2. Augmentation If  $A \rightarrow B$ , then  $A, C \rightarrow B, C$
3. Transitivity If  $A \rightarrow B$  and  $B \rightarrow C$ , then  $A \rightarrow C$

## Data Base Management Systems

### The Process of Normalization

- ◆ Formal technique for analyzing a relation based on its primary key and functional dependencies between its attributes.
- ◆ Often executed as a series of steps. Each step corresponds to a specific normal form, which has known properties.
- ◆ As normalization proceeds, relations become progressively more restricted (stronger) in format and also less vulnerable to update anomalies.

### Relationship Between Normal Forms



### Unnormalized Form (UNF)

- ◆ A table that contains one or more repeating groups.
- ◆ To create an unnormalized table:
  - transform data from information source (e.g. form) into table format with columns and rows.

### First Normal Form (1NF)

- ◆ A relation in which intersection of each row and column contains one and only one value.

### UNF to 1NF

- ◆ Nominate an attribute or group of attributes to act as the key for the unnormalized table.

## Data Base Management Systems

- ◆ Identify repeating group(s) in unnormalized table which repeats for the key attribute(s).
- ◆ Remove repeating group by:
  - entering appropriate data into the empty columns of rows containing repeating data ('flattening' the table).
 Or by
  - placing repeating data along with copy of the original key attribute(s) into a separate relation.

### Second Normal Form (2NF)

- ◆ Based on concept of full functional dependency:
  - A and B are attributes of a relation,
  - B is fully dependent on A if B is functionally dependent on A but not on any proper subset of A.
- ◆ 2NF - A relation that is in 1NF and every non-primary-key attribute is fully functionally dependent on the primary key.

### 1NF to 2NF

- ◆ Identify primary key for the 1NF relation.
- ◆ Identify functional dependencies in the relation.
- ◆ If partial dependencies exist on the primary key remove them by placing them in a new relation along with copy of their determinant

### Third Normal Form (3NF)

- ◆ Based on concept of transitive dependency:
  - A, B and C are attributes of a relation such that if  $A \twoheadrightarrow B$  and  $B \twoheadrightarrow C$ ,
  - then C is transitively dependent on A through B. (Provided that A is not functionally dependent on B or C).
- ◆ 3NF - A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on the primary key.

### 2NF to 3NF

- ◆ Identify the primary key in the 2NF relation.
- ◆ Identify functional dependencies in the relation.

## Data Base Management Systems

- ◆ If transitive dependencies exist on the primary key remove them by placing them in a new relation along with copy of their determinant.

### General Definitions of 2NF and 3NF

- ◆ Second normal form (2NF)
  - A relation that is in 1NF and every non-primary-key attribute is fully functionally dependent on any candidate key.
- ◆ Third normal form (3NF)
  - A relation that is in 1NF and 2NF and in which no non-primary-key attribute is transitively dependent on any candidate key.

### Boyce-Codd Normal Form (BCNF)

- ◆ Based on functional dependencies that take into account all candidate keys in a relation, however BCNF also has additional constraints compared with general definition of 3NF.
- ◆ BCNF - A relation is in BCNF if and only if every determinant is a candidate key.
- ◆ Difference between 3NF and BCNF is that for a functional dependency  $A \rightarrow B$ , 3NF allows this dependency in a relation if B is a primary-key attribute and A is not a candidate key.
- ◆ Whereas, BCNF insists that for this dependency to remain in a relation, A must be a candidate key.
- ◆ Every relation in BCNF is also in 3NF. However, relation in 3NF may not be in BCNF.
- ◆ Violation of BCNF is quite rare.
- ◆ Potential to violate BCNF may occur in a relation that:
  - contains two (or more) composite candidate keys;
  - the candidate keys overlap (i.e. have at least one attribute in common).

### Review of Normalization (UNF to BCNF)

## Data Base Management Systems

DreamHome Property Inspection Report					
DreamHome Property Inspection Report					
Property Number <u>PG4</u>					
Property Address <u>6 Lawrence St, Glasgow</u>					
Inspection Date	Inspection Time	Comments	Staff no	Staff Name	Car Registration
18-Oct-00	10.00	Need to replace crockery	SG37	Ann Beech	M231 JGR
22-Apr-01	09.00	In good order	SG14	David Ford	M533 HDR
1-Oct-01	12.00	Damp rot in bathroom	SG14	David Ford	N721 HFR

Page 1

### StaffPropertyInspection

propertyNo	pAddress	iDate	iTime	comments	staffNo	sName	carReg
PG4	6 Lawrence St, Glasgow	18-Oct-00	10.00	Need to replace crockery	SG37	Ann Beech	M231 JGR
		22-Apr-01	09.00	In good order	SG14	David Ford	M533 HDR
		1-Oct-01	12.00	Damp rot in bathroom	SG14	David Ford	N721 HFR
PG16	5 Novar Dr, Glasgow	22-Apr-01	13.00	Replace living room carpet	SG14	David Ford	M533 HDR
		24-Oct-01	14.00	Good condition	SG37	Ann Beech	N721 HFR

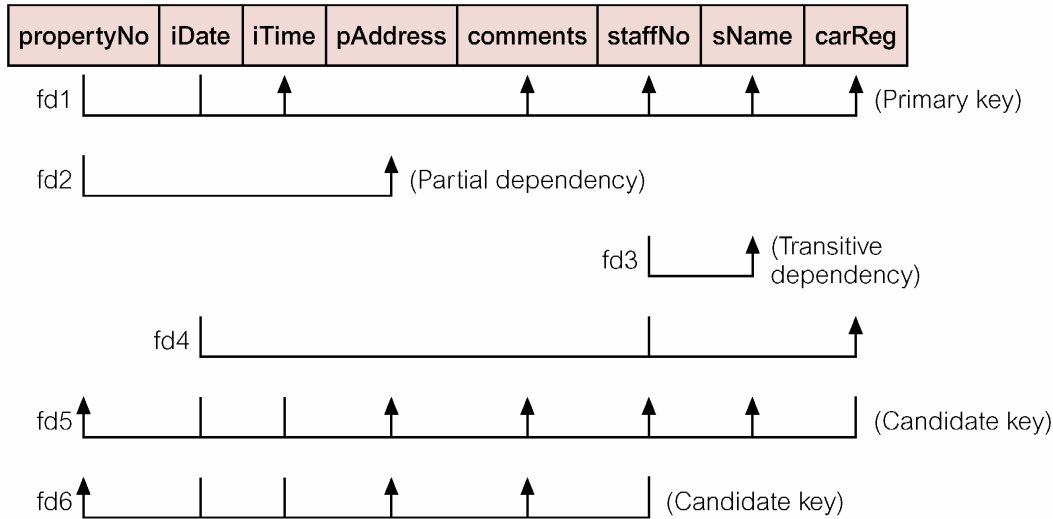
### StaffPropertyInspection

propertyNo	iDate	iTime	pAddress	comments	staffNo	sName	carReg
PG4	18-Oct-00	10.00	6 Lawrence St, Glasgow	Need to replace crockery	SG37	Ann Beech	M231 JGR
PG4	22-Apr-01	09.00	6 Lawrence St, Glasgow	In good order	SG14	David Ford	M533 HDR
PG4	1-Oct-01	12.00	6 Lawrence St, Glasgow	Damp rot in bathroom	SG14	David Ford	N721 HFR
PG16	22-Apr-01	13.00	5 Novar Dr, Glasgow	Replace living room carpet	SG14	David Ford	M533 HDR
PG16	24-Oct-01	14.00	5 Novar Dr, Glasgow	Good condition	SG37	Ann Beech	N721 HFR

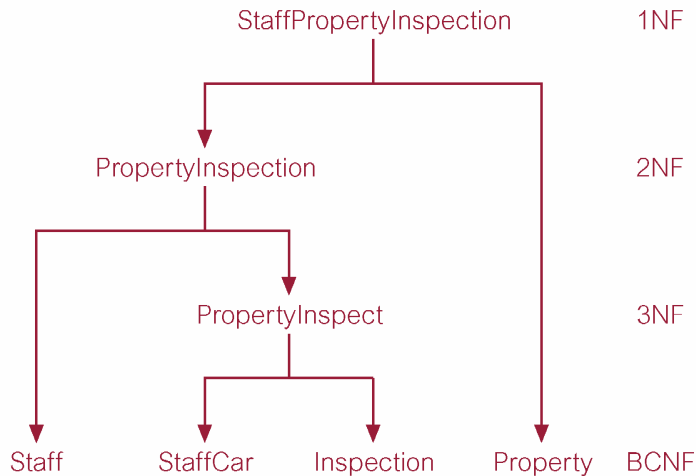
### Review of Normalization (UNF to BCNF)

## Data Base Management Systems

### StaffPropertyInspection



### Review of Normalization (UNF to BCNF)



### Fourth Normal Form (4NF)

- ◆ Although BCNF removes anomalies due to functional dependencies, another type of dependency called a multi-valued dependency (MVD) can also cause data redundancy.
- ◆ Possible existence of MVDs in a relation is due to 1NF and can result in data redundancy.
- ◆ Dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B and a set of values for C. However, set of values for B and C are independent of each other.



## Data Base Management Systems

- ◆ MVD between attributes A, B, and C in a relation using the following notation:  
 $A \twoheadrightarrow B$   
 $A \twoheadrightarrow C$
- ◆ MVD can be further defined as being trivial or nontrivial.
  - MVD  $A \twoheadrightarrow B$  in relation R is defined as being trivial if (a) B is a subset of A or (b)  $A \cup B = R$ .
  - MVD is defined as being nontrivial if neither (a) nor (b) are satisfied.
  - Trivial MVD does not specify a constraint on a relation, while a nontrivial MVD does specify a constraint.
- ◆ **Defined as a relation that is in BCNF and contains no nontrivial MVDs.**

### 4NF - Example

BranchStaffOwner

branchNo	sName	oName
B003	Ann Beech	Carol Farrel
B003	David Ford	Carol Farrel
B003	Ann Beech	Tina Murphy
B003	David Ford	Tina Murphy

BranchStaff

branchNo	sName
B003	Ann Beech
B003	David Ford

BranchOwner

branchNo	oName
B003	Carol Farrel
B003	Tina Murphy

### Fifth Normal Form (5NF)

- ◆ A relation decomposed into two relations must have lossless-join property, which ensures that no spurious tuples are generated when relations are reunited through a natural join.
- ◆ However, there are requirements to decompose a relation into more than two relations.
- ◆ Although rare, these cases are managed by join dependency and fifth normal form (5NF).
- ◆ **A relation that has no join dependency.**

## Data Base Management Systems

(a) PropertyItemSupplier (Illegal state)

propertyNo	itemDescription	supplierNo
PG4	Bed	S1
PG4	Chair	S2
PG16	Bed	S2

When this tuple is added to relation.

(b) PropertyItemSupplier (Legal state)

propertyNo	itemDescription	supplierNo
PG4	Bed	S1
PG4	Chair	S2
PG16	Bed	S2
PG4	Bed	S2

This new tuple must also be added to exist in any legal state of the relation.

### 5NF - Example

PropertyItem

propertyNo	itemDescription
PG4	Bed
PG4	Chair
PG16	Bed

ItemSupplier

itemDescription	supplierNo
Bed	S1
Chair	S2
Bed	S2

PropertySupplier

propertyNo	supplierNo
PG4	S1
PG4	S2
PG16	S2

More Notes:

### Inference Rules for FD

- **Reflexive Rule :**  
if  $X \leq Y$ , then  $X \rightarrow Y$
- **Augmentation Rule :**  
 $\{X \rightarrow Y\} = XZ \rightarrow$
- **Transitive Rule:**  
 $\{X \rightarrow Y, Y \rightarrow Z\} = X \rightarrow Z$
- **Decomposition or Projective Rule**  
 $\{X \rightarrow YZ\} = X \rightarrow Y$
- **Union or additive Rule :**  
 $\{X \rightarrow Y, X \rightarrow Z\} = X \rightarrow YZ$
- **Pseudo Transitive Rule:**  
 $\{X \rightarrow Y, WY \rightarrow Z\} = WX \rightarrow Z$

Minimal sets of FD

## Data Base Management Systems

- **Algorithm:** Finding minimal cover F for a set of Functional Dependencies E

  1. set  $F := E$
  2. Replace each FD  $X \rightarrow \{A_1, A_2, \dots, A_n\}$  in F by the n FD  $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_n$
  3. For each FD  $X \rightarrow A$  in F
    - for each attribute B that is an element of X
    - if  $\{F - \{X \rightarrow A\}\} \cup \{(X - \{B\}) \rightarrow A\}$  is equivalent to F then
    - replace  $X \rightarrow A$  with  $(X - \{B\}) \rightarrow A$  in F
  4. for each FD  $X \rightarrow A$  in F
    - if  $\{F - \{X \rightarrow A\}\}$  is equivalent to F then remove  $X \rightarrow A$  from F
  
- **First Normal Form:**

  - A relation is said to be in First NF if it is already in un normalized form and it has no repetitive group
- **Second NF**

  - A relation is said to be in second NF if it is already in the First NF and it has no partial dependency
- **Third NF**

  - A relation is said to be in Third NF if it is already in second NF and it has no transitive dependency
- **BCNF**

  - A relation is said to be in Boyce Codd NF if it is already in the third NF and every determinant is a candidate key. It is a stronger version of the third NF.
- **Fourth NF**

  - A relation is said to be in Fourth NF if it is already in BCNF and it has no multi valued dependency
  - **Multi Valued Dependency**
    - Consider three fields X, Y and Z in a relation. If for each value of X, there is a well defined set of values Y and well defined set of values Z and the set of values of Y is independent of the set of values of Z, then multi valued dependency exists.
- **Fifth NF**

  - A relation is said to be in Fifth NF if it is already in fourth NF and it has no join dependency

## Data Base Management Systems

**Note: This chapter only for Bsc.IT**

### Introduction to PL/SQL

PL/SQL stands for procedural language extensions to SQL. It is available as an enabling technology within other Software Products; It does not exist as a stand – alone language with PL/SQL, we can use SQL statements to manipulate oracle data and flow of control statements to process the data. Also we can declare constants and variables defined in subprograms and trap runtime errors. Thus PL/SQL combines the data manipulating power of SQL with the data processing power of procedural languages.

#### Why Use PL/SQL?

PL/SQL is a complete transaction processing language that provides the following advantages:

1. It acts as a support for SQL. Since it supports SQL data manipulation statement, SQL transaction processing statement, SQL functions and SQL predicates we can access the ORACLE database and manipulate its data flexibly and easily.
2. PL/SQL allows to declare variables and constants and use them in SQL and procedural statements, any where an expression can be used. PL/SQL variables and constants have attributes (%Type, %Rowtype) by which we can reference the data type and structure of an object without repeating its definition.
3. Control structures are the most important PL/SQL extension of SQL. It allows you process the data using the conditional control statement(IF-THEN-ELSE), iterative control statement (LOOP END LOOP, FOR-LOOP and WHILE-LOOP) and sequential control statement (GOTO), with the help of these statements we can handle any situation.
4. A PL/SQL cursor gives you a way to fetch and process database information. There are two types of cursors EXPLICIT and IMPLICIT CURSORS. PL/SQL declares cursor implicitly for all SQL manipulation statements including queries that returns one row. FOR queries that returns more than one row an explicit cursor has to be declared.
5. Modularity is promoted because PL/SQL lets you break an application down into manageable, well-defined logic modules.
6. Errors are easily detected and handled.

#### What Commands Can Be Executed From SQL\*Plus?

You can enter three kinds of commands from the SQL\*Plus command prompt:

SQL\*Plus commands – for formatting query results, setting options, and editing and storing SQL commands and PL/SQL blocks. Eg.

## Data Base Management Systems

SQL commands – like select, create, alter, drop etc. Eg:

```
SELECT*FROM TAB;
```

PL/SQL blocks – a set of line that do a particular task. Eg:

```
BEGIN
DBMS_OUTPUT.PUT_LINE('Hello World');
END;
/
```

### SQL\*Plus Basics

Oracle's SQL\*Plus is a command line tool that allows a user to type SQL statements to be executed directly against an Oracle database. SQL\*Plus has the ability to format database output, save often used commands and can be invoked from other Oracle tools or from the operating system prompt.

### PL/Sql Block Structure

PL/SQL is a block structured language. A block lets you group logically related declarations and statements. The declarations are local to the block and cease to exist when the block completes. A block has three parts: a declarative part, and executable part, and an exception handling part (error handling).

```
DECLARE
<declaration>
BEGIN
<executable statements>
EXCEPTION
<exception handlers>
END;
```

The declarative part comes first in which the objects can be declared. Once declared the objects can be manipulated in the executable part. Exceptions raised during execution can be handled in the exception-handling part. We can nest sub-blocks in the executable, exception part and not in declarative part. You can define subprograms (Functions, Procedures) in the declarative part of any block. But it can be called with in the block only. See example:

```
DECLARE
<declaration>
BEGIN
SUB-BLOCK1
DECLARE
<declarations>
BEGIN
```

## Data Base Management Systems

```

EXCEPTION
<Exception handlers>
END;
SUB-BLOCK2
DECLARE
<declarations>
BEGIN
<executable statements>
<EXCEPTION>
<exception handlers>
END;
END;

```

Referring to the following sample PL/SQL block we have the following example.

```

DECLARE
Num_in_stock NUMBER(5);
BEGIN
    SELECT quantity INTO num-in-stock FROM inventory_table
    WHERE product = 'TENNIS RACQUET';

IF num_in_stock>0 THEN

    UPDATE INVENTORY_TABLE SET quantity = quantity - 1
    WHERE product = 'THNNIS RECQUET';

    INSERT INTO purchase_record
    VALUES ('TENNIS RACQUET PURCHASED',, SYSDATA);
ELSE

    INSERT INTO purchase_record
    VALUES('OUT OF TENNIS RACQUETS.',SYSDATA);

END IF;
COMMIT;
END;

```

### Advantages of PL/SQL

PL/SQL is a completely portable high performance transaction processing language which provides the following advantages:

#### Support for Sql:-

PL/SQL lets us to use all SQL data manipulation, cursor control and transaction control commands as well as SQL functions, operators, and pseudocolumns.

#### Higher Productivity:-

PL/SQL adds functionality to non-procedural tools such as SQL\*Forms, SQL\*

## Data Base Management Systems

construct to build applications. Thus it increases productivity by putting better tools in your hands.

### Better Performance:-

With our PL/SQL Oracle must process SQL statements one at a time. It increases network traffic. But with PL/SQL an entire block of SQL statements (more than one) can be sent to Oracle at one time. Also with the procedural capabilities in enhances performance.

### Portability:-

Applications written in PL/SQL are portable to any operating system and platform in which Oracle runs.

### Character Set

A PL/SQL program consists of sequence of statements, each of which is made up of one or more lines of text. Text is made up of combinations of the characters shown below:

- ❖ The numerals 0-9
- ❖ Tab, space, carriage return
- ❖ The symbols ( ) + - \* / <> = ; : @ % , " # \$ ^ & \_ ! { } [ ]
- ❖ The upper and lower-case letters A.....Z, a.....z.

PL/SQL is not case sensitive, so lower-case letters are equivalent to the corresponding upper-case letters except within string (when surrounded by single quotes) or represent the value of a character variable.

### Delimiters:-

A delimiters is a simple or compound symbol that has a special meaning to PL/SQL. A few examples are given below.

- ;  
statement terminator
- %  
Attribute indicator (cursor attributes like % is open and indirect declaration attributes like % Rowtype
- Single underscore: (single byte wildcard symbol as in SQL.
- :  
Host variable indicator  
<<and>>Label delimiters
- :=  
Alignment operator
- Single line comment
- /\* and \*/  
Beginning and ending multiline comment block delimiters.

### Identifiers:-

An identifier is a name for a PL/SQL object which includes constants, variables, exceptions, cursors, subprograms, packages, and reserved words. The properties of identifiers are:

## Data Base Management Systems

1. Upto 30 characters in length
2. Must start with a letter
3. Can include \$ sign, underscore, and #(pound sign)
4. Cannot contain spaces.

### Binary Integer:-

It allows to store signed integers ( $-2^{31}-1$  through  $2^{31}-1$ ). Natural and positive are both subtypes of Binary-integer

Natural allows 0 through  $2^{31}-1$

Positive allows 1 through  $2^{31}-1$

### Number

Use the number data type to store fixed or floating point numbers of any size. The maximum precision of a variable with Number type is 38 digits. While declaring we can optionally specify the valuable precision as follows:

Number (precision, Scale)

The precision of a Number is the total number of digits. The scale dictates the number of digits to the right or left of the decimal point at which rounding occurs. The remaining data types in number are all sub types of number. They have the same range of values as their base type. The remaining data types in number are all sub types of number. They have the same range of values as their base type.

Subtype	Datatype
Dec(prec.scale)	Number(prec, Scale)
Decimal(prec.scale)	Number(prec.scale)
Double Precision	Number
Float (Binary)	Number
Int	Number(38)
Integer	Number(38)
Numeric (Prec, Scales)	Number(prec, Scale)
Real	Number
Smallint	Number(38)

### Char

Variables with character data types store text and are manipulated by character functions. The CHAR data type takes an optional parameter that lets you specify a maximum length upto 32767 bytes. The syntax is:-

CHAR (maximum\_length)

### Char Subtypes

The character subtypes have the same range of values as their base type.



## Data Base Management Systems

### Character

#### Varchar 2

It has the same syntax as Char. But the difference is, if we assign a character value to a Char variable and if the length is shorter than the declared length of the variable PL/SQL blankpads the value to the declared length. The VARCHAR is a subset of VARCHAR2.

If we assign a character value to a Varchar2 variable, and if the value is shorter than the declared length of the variable, PL/SQL neither blankpads the value nor strips trailing blanks.

```
Name1 Char (10);
Name2 Varchar2(10);
```

If we give name1 and name2 as Frank, name 1 will be stored internally as 'Frank' and name2 as 'Frank'. So even though the names are same since the data types are different they are not same.

#### Long:-

The variable LONG can store variable-length strings of up to 32760 bytes which is seven fewer bytes longer than allowed in Varchar2 type variables.

#### Date:-

The Date data type is to store fixed length data values. It also takes no parameters. Valid dates include from Jan 1,4712 BC to December 31,4712 AD. When stored in a database column, date values include to the first day of the current month; the time portion defaults to midnight.

#### Declare Variable:-

Variable are declared in the DECLARE section of the PL/SQL block. Declaration involves the name of the variable followed by its data type. All statements must end with a semicolon. Initial values can also be assigned at the time of declaration. Constants are declared by specifying the key work CONSTANT before the data type. The syntax is:-

**Identifier [Constant] Data Type [Not Null] [:=Pl/Sql Expression];**

#### Example:

Declare	Number
Count	Number (9,2)
Secs_Per_Day Constant	Number:= 60*60*24
First_Name	Char
Last Name	Varchar2
Birth_Day	Date
Available	Boolean:= Null;

## Data Base Management Systems

.....  
 .....

### Comparisons

PL/SQL supports the comparison of variables and constants in SQL and PL/SQL statements. These comparisons called Boolean expressions, generally consists of simple expressions separated by Relational operators (=,!=,<,>, >=, <=). Boolean expressions are often connected by the logical operators AND, OR, NOT. In PL/SQL a Boolean expression always evaluates to TRUE, FALSE OR NULL. In a SQL statement Boolean expression allow you to control which rows in a table are affected by the statement. In a non SQL statement, Boolean expressions are the basis for conditional control.

There are three kinds of Boolean expression Numeric, Character, Date.

#### Boolean Expression:-

A character string is a sequence of characters stung together. Like numeric expressions, character string can be compared. The comparison is based on the alphabetic ordering.

```
ename>empname
ename != 'frank'
```

Operator	Meaning
=	Is the Same As
!=	Is Not The Same As
<	Comes Alphabetically before
>	Comes Alphabetically After
<=	Comes Alphabetical Before

#### Declaring Variables and Constants:-

PL/SQL supports a variety of data types that you can use for declaring variables and constants. You can assign values to variables as you declare them and you can change the value of a variable through further assignments.

You must assign the value of a constant to it when you declare it, this value is fixed and cannot be changed at run time.

Example variable declarations are given below:-

```
DECLARE
    V_NUM1 NUMBER NOT NULL:=10109;
    NUM8 NUMBER(3,1);
    XYZ NUMBER(2,2):=31.8;
    ABC12 NUMBER(9,2):=XYZ*131;
    V_CHR1 CHAR(89);
    V_CHR2 VARCHAR2(12):= "JANUARY";
    TODAY DATE:= SYSDATE;
```

## Data Base Management Systems

Example constant declarations are given below:-

**DECLARE**

```
PI CONSTANT NUMBER(9,3) : 3.142;  
VAT CONSTANT NUMBER(4,2):=17.5;
```

When you declare PL/SQL variables to hold column values you must ensure that the variable type is the same as the column type else you will get a run time error at execution. You can use the %TYPE attribute to base a variable upon the column definition as defined within the Oracle data dictionary. The attribute is prefixed with the schema, table and column name and used where the datatype is required in the DECLARE SECTION. PL/SQL determines the datatype and size of the variable when the block is compiled and so is always compatible with the column used to populate it. An example follows:-

**DECLARE**

```
V_NUM1 JD11.BOOK.COST%TYPE;
```

**PL/SQL Editor:-**

PL/SQL Editor is nothing but where you type all the pl/sql programs. In the following sections we see how to write different types of programs using the standard structures available in PL/SQL.

**PL/SQL Structures**

Using SQL statements, we can retrieve or manipulate data present in a table. Using SQL statements above it is not possible to gain the power of procedural language constructs. This aspect has been taken care of by PL/SQL, which is of a procedural extension of SQL.

A PL/SQL block can contain DML & DCL statements, but not DDL statements – but not DDL statements. A PL/SQL block can also contain any number of SQL statements integrated with flow of control statements. Using PL/SQL we can also trap runtime errors.

**What are Pl/Sql structures?**

With PL/SQL, you can use SQL statements to manipulate Oracle data and flow control statements to process data to meet all your requirements. PL/SQL combines the data manipulating power of SQL with the data-processing power of procedural languages. PL/SQL provides many different structures to control the flow of your statement execution and data handling.

**Why use Pl/Sql Structures?**

## Data Base Management Systems

Control structures are the most important PL/SQL extensions to SQL. Not only does PL/SQL let you manipulate data, it lets you process the data using conditional, iterative, sequential, and unconditional flow-control statements such as if-then-else, for loop, while-loop, exit-when and goto.

### Control Structures:-

There are three types of control statements. Conditional control, Iterative control and Sequential control. The conditional control returns a Boolean value. The iterative control executes a sequence of statements repeatedly. The sequential statements execute a sequence of statements in the order in which they occur.

### Conditional Control:-

The IF statement lets you execute a series of statements conditionally. There are three forms IF-THEN, IF-THEN-ELSE, and IF-THEN-ELSIF.

#### If-then

IF associates a condition with a sequence of statements enclosed by the keywords THEN and END IF. The statements are executed only when the condition is TRUE.

IF condition THEN

```

    Sequence of statements;
END IF;
```

#### Example:

```

IF Sales>Quata THEN
    Combute_bonus(empid);
    UPDATE payroll SET pay + bonus WHERE EMPNO = EMP_ID;
END IF;
```

#### If-then-else

The second form of IF statements adds the keywords ELSE followed by alternative sequence of statements. The sequence of statements in the ELSE clause will be executed only when the condition evaluates to FALSE.

IF condition THEN

```

    Sequence of statement
ELSE
    Sequence of statements
END IF;
```

## Data Base Management Systems

**Example:-**

```

IF trans_type = 'CR' THEN
    UPDATE accounts SET balance = balance + credit WHERE.....
ELSE
    UPDATE accounts SET balance = balance - debit WHERE
END IF;

```

The third form introduces the keyword ELSIF to introduce additional conditions. If any condition evaluates to TRUE then that sequence of statements get executed. We can have any number of ELSIF clause but the last one is optional.

```

If conditiona 1 THEN
    Sequence of statements;
ELSIF
    Sequence of statements;
ELSIF
    Sequence of statements;
END IF;

```

An example of if-then-else structure is as follows.

```

SQL>declare
2 length number(3):=10;
3 breadth number(3):=12;
4 area number(5);
5 begin
6 if length>=0 then
7 if breadth <=0 then
8 dbms_output.put_line('Breadth cant be less than 0');
9 else
10 area:=.5 * length * breadth;
11 dbms_output.put_line('Breadth cant be less than 0');
12 end if;
13 else
14 dbms_output.put_line('Length cant be less than 0');
15 end if;
16 end;
17/

```

Area is 60  
PL/SQL procedure successfully completed.

### Iterative Control

Loop statements lets you use the iterative type of control. There are three forms of LOOP statements. LOOP, WHILE-LOOP and FOR-LOOP.

**Loop**

## Data Base Management Systems

The simplest form is the infinite LOOP, WHICH encloses sequence of statements between the key words LOOP and END LOOP.

### LOOP

Sequence of statements;  
END LOOP;

### Exit

The EXIT statement is used to complete the loop. You can place one or more EXIT statement inside a LOOP. The EXIT-WHEN statement allows a loop to complete conditionally when the condition in the WHEN clause is evaluated.

### Loop

If..then

    Exit; - Exit Loop

End If;

End Loop;

### Loop

Fetch C1 Into ...

    Exit When C1% Notfound...

End Loop;

### Example

SQL>declare

2 counter binary\_integer:=1;

3 begin

4 loop

5 dbms\_output.put\_line('Counter values is' || counter);

6 demb\_output.put\_line(“);

7 counter:=counter+1;

8 exit when counter>5;

9 end loop;

10 end;

/

Counter values is 1

Counter values is 2

Counter values is 2

Counter values is 3

Counter values is 5

PL/SQL procedure successfully completed.

### Loop Labels

Like PL/SQL blocks LOOPS can be labeled. The label, an undeclared identifier enclsd by double angle brackets, must appear at the beginning of the LOOP statement as follows:

## Data Base Management Systems

### Loop

```
.....
End Loop Label_Name;
<<Outer Block>>
```

### While-loop

The WHILE-LOOP statement associates a condition with a sequence of statement enclosed by the keywords LOOP and END LOOP. Before each iteration of the LOOP, the condition is evaluated. If the condition evaluates to TRUE the sequence of statements are evaluated else the loop is by passed.

### While Condition Loop

```
Sequence Of Statements:
Endloop;
```

Consider the following example. This block uses a simple WHILE loop to insert 5 rows into a table. The values of a counter variable, and either of two character strings are inserted. Which string is inserted depends on the value of the loop index.

```
<SQL> create table TEMP
2 (coll number(9,4));
```

Table created.

```
<SQL>DECLARE
2 x NUMBER:= 100;
3 BEGIN
4 WHILE x<501 LOOP
5 INSERT INTO TEMP VALUES (X);
6 x:= x+100;
7 END LOOP;
8 COMMIT;
9 END;
10 /
```

PL/SQL procedure successfully completed.

### For-loop

The number of iterations through a WHILE-LOOP is unknown. The number of iterates through a FOR-LOOP is known before it enters the LOOP, FOR-LOOP iterates over a specified range of integers. The syntax is as follows:

For example consider the following example which will check the value I is even or odd and inserts the data accordingly.

```
SQL>create table TEMP
2 (col1 number(9,4),
3 (col2 number(4),
```

## Data Base Management Systems

### Table created

```
SQL>DECLARE
2 x NUMBER:=100;
3 BEGIN
4 FOR I IN 1..5 LOOP
5 IF MOD(I,2)=0 THEN – I is even
6 INSERT INTO temp VALUES(i,x, 'i(even)');
7 ELSE
8 INSERT INTO temp VALUES(i,x, 'i(odd)');
9 END IF;
10 x:=x+100;
11 END LOOP;
12 COMMIT;
13 END;
14/
```

PL/SQL procedure successfully completed.

### Loop

Insert Into Temp (Coll.message) Values (Count, 'Ever Green');  
Insert Ten Times Is Ever Green Into Temp End Loop.

### Sequential Control

GOTO and NULL statements are the sequential control statements.

### Goto

The GOTO statement branches to a label unconditionally. The label must be unique within its scope and must precede an executable statement or a PL/SQL block. When executed, the GOTO statement transfers control to the labeled statement or block. The syntax is below:-

```
BEGIN
.....
GOTO test_line;
.....
<<test line>>
.....
END;
```

- ❖ A GOTO can't branch into an IF statement, LOOP statement or sub block.
- ❖ Also it can't branch from one IF statement clause to another.
- ❖ A GOTO cant branch out of a subprogram.
- ❖ A GOTO statement cant branch from an exception handler into the current block.



## Data Base Management Systems

### NULL STATEMENT

The NULL statement specified inaction, it does nothing other than pass control to the next statement. It can, however, improve readability. In a construct allowing alternative actions, the NULL statement serves as a place holder.

```
If  
Score>100 Then  
Compute_Bonus(Batsma_Id);  
Else  
Null;  
End If;
```

### Cursors

A cursor is a variable that runs through the tuples of some relation. This relation can be a stored table, or it can be the answer to some query. By fetching into the cursor each tuple of the relation, we can write a program to read and process the value of each such tuple. If the relation is stored, we can also update or delete the tuple at the current cursor position.

To process a SQL statement, PL/SQL opens work areas called private SQL area. PL/SQL allows user to name the private work areas and access the stored information. The PL/SQL construct to identify each and every work area used is called CURSOR. There are two types of cursors namely IMPLICIT and EXPLICIT.

### Explicit Cursor

The set of rows returned by a query can consist of Zero, one or many rows, depending upon the number of rows that meet the query's search condition. When a query returns multiple rows, a cursor can be explicitly defined to:

- 1) process beyond the first row returned by the query.
- 2) Keep track of which row is being processed.

There are four steps to DECLARE and USE A CURSOR:

```
DECLARE the cursor  
Open the cursor  
FETCH data from the cursor  
CLOSE the cursor
```

### Declare Cursor

Declare the cursor to associate its name with a select statement. Forward references are not allowed in PL/SQL. So we must declare cursor before referencing in other statements.

### DECLARE

## Data Base Management Systems

< select statement>:

**Example:**

```
CURSOR MY_FIRST IS SELECT ENAME FROM EMP
WHERE SAL>7000;
```

- The select statement must not include the INTO clause
- Values can be assigned to cursor name
- Declared cursors are scoped just as variables

**Open Cursor**

Opening the cursor executes the query and identifies the active set which consists of all rows that meet the query search criteria.

```
OPEN<cursor_name>;
```

**Fetch Cursor**

The fetch statement retrieves the rows in the active set one at a time. Each time the fetch is executed the cursor advances to the next row in the active set.

```
FETCH<cursor_name>INTO<var, var2..>;
```

For each column value returned by the query associated with the cursor, there must be a corresponding variable in the INTO list. Their datatypes must also be compatible.

**Close Cursor**

Close the cursor to free up the resources. The close statement disables the cursor and the active set becomes undefined. No more rows can be fetched from a closed cursor.

```
CLOSE<cursor_name>;
```

**Example**

```
SQL>DECLARE CURSOR c1 IS
2 SELECT ename, sal FROM emp;
3 name emp.ename%type;
4 salary emp.sal%type;
5 BEGIN
6 OPEN
7 for I in 1..5
8 LOOP
9 FETCH c1 INTO name, salary;
10 dbms_output.put_line('Name is' || name || 'Salary is' || salary);
11 end loop;
12 close c1;
13 end;
14/
```

## Data Base Management Systems

Name is ALLEN Salary is 1600  
 Name is WARD Salary is 1250  
 Name is JONES Salary is 2975  
 Name is SMITH Salary is 1250

PL/SQL procedure successfully completed.

```

1) DECLARE
    /* Output variables to hold the result of the query:*/
2) a T1.e%TYPE;
3) b T1.f%TYPE;
    /*cursor declaration:*/
4) CURSOR T1Cursor IS
5) SELECT e,f
6) FROM T1
7) WHERE e<f
8) FOR UPDATE;
9) BEGIN
10) OPEN T1Cursor;
11) LOOP
    /* retrieve each row of the result of the above query
into PL/SQL variables:*/
12) FETCH T1Cursor INTO a,b;
    /* If there are no more rows to fetch, exit the loop:*/
13) EXIT WHEN T1 CURSOR%NOTFOUND;
14) /DELETE FROM T1 WHERE CURRENT OF T1Cursor;
    /* Insert the reverse tuple:*/
15) INSERT INTO T1 VALUES (b,a);
16) END LOOP;
    /*Free cursor used by the query.*/
17) CLOSE T1 Cursor;
18) END;
19) .
20) run;
  
```

### Subprograms

The PL/SQL programs can be stored in the database as stored programs and can be invoked whenever required. This avoids repassing when multiple users invoke it. This also provides security and integrity control by allowing access on the sub program and not on the database objects directly. Actually subprograms are named PL/SQL blocks that can take parameters and be invoked.

### What Is A Subprogram?

Subprograms are PL/SQL blocks that can take parameter and can be invoked.

PL/SQL has two types of subprograms PROCEDURES and FUNCTIONS. A

## Data Base Management Systems

### Why Use Subprograms?

- Provides extensibility
- Provides modularity
- Promotes reusability and manageability

### Using Subprograms

Subprograms have a declarative part, an executable part and an optional exception handling part. The declarative part contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These objects are local and cease to exist when exited from the subprogram. The executable part contains statements that assign values, control execution, and manipulate ORACLE data. The exception handling part contains exception handlers, which deal with exceptions raised during exception.

A procedure is a subprogram that performs a specific action. The syntax is:

```
PROCEDURE name [parameter [,parameter]] IS local declaration
BEGIN
```

Executable statements

```
[EXCEPTIONS]
```

```
END [name];
```

Where parameter stands for

```
Var_name [IN/OUT/IN OUT] datatype
```

```
[{:=!DEFAULT} value]
```

A procedure has two parts the SPECIFICATION and the BODY. A specification begins with the keyword PROCEDURE and ends with the procedure name or parameter list. A body begins with the keywords IS and ends with the keyword END. A body has three parts a DECLARATIVE PART, an EXECUTABLE part, and an optional EXCEPTION handling part.

PL/SQL procedures behave very much like procedures in other programming language. Here is an example of a PL/SQL PROCEDURE addtuple1 that, given an integer I, inserts the tuple (i, 'xxx') into the following example relation:

```
CREATE TABLE T2 (
                a INTEGER,
                b CHAR(10)
CREATE Procedure addtuple1 (i IN NUMBER) AS
BEGIN
    INSERT INTO T2 VALUES(i, 'xxx');
END addtuple1;
```

A procedure is introduced by the keywords CREATE PROCEDURE followed by the procedure name and its parameters. An option is to follow CREATE by OR REPLACE. The advantage of doing so is that should you have already made the definition, you will not get an error. On the other hand, should the previous definition

## Data Base Management Systems

be a different procedure of the same name, you will not be warned, and the old procedure will be lost.

There can any number of parameters, each followed by a mode and a type. The possible modes are IN (read-only), OUT(write-only), and INOUT(read and write).

```
BEGIN
Addtyle1(99);
END;
```

The following procedure also inserts a tuple into T2, but it takes both components as arguments:

```
CREATE PROCEDURE addtuple2(
                                xT2,a%TYPE
                                y T2,b%TYPE)
AS
BEGIN
INSERT INTO T2(A,B)
VALUES(x,y);
END addtuple2;
```

Now, to add a tuple2(10, 'abc') to T2:

```
BEGIN
    Addtyle2(10, 'abc');
END;
```

The following illustrates the use of an OUT parameter:

```
CREATE TABLE T3 (
    a INTEGER,
    b INTEGER
);

CREATE PROCEDURE addtuple3(a NUMBER, b OUT NUMBER)
AS
BEGIN
    B :=4;
    INSERT INTO T3 VALUES(A,B);
END;
DECLARE
v NUMBER;
BEGIN addtuple3(10,v);
END;
```

Note that assigning values to parameters declared as OUT or INOUT causes the corresponding input arguments to be written. Because of this, the input argument for an OUT or INOUT parameter should be something with an "lvaluse", such as a variable like v in the example above. A constant or a literal argument should not be passed in for an OUT/INOUT parameter.

## Data Base Management Systems

```

SQL>CREATE OR REPLACE PROCEDURE MYPROC1
2 IS
3 TEMP_SAL NUMBER;
4 BEGIN
5 SELECT SAL INTO TEMP_SAL FROM EMP WHERE EMPNO = 7369;
6 IF TEMP_SAL>0 THEN
7 UPDATE EMP SET SAL = (TEMP_SAL+200) WHERE EMPNO = 7369;
8 ELSE
9 UPDATE EMP SET SAL = 200 WHERE EMPNO = 7369;
10 END IF;
11 COMMIT;
12 EXCEPTION
13 WHEN NO_DATA_FOUND THEN
14 INSERT INTO ERR(CODE, MESSAGE) VALUES(1, 'EMPLOYEE 7369 NOT
FOUND');
15 END MYPROC1;

```

Procedure created.

The procedure named MYPROC1 is defined in the first line. CREATE OR REPLACE asks Oracle to create a new procedure or if a procedure with this name already exists in this schema to replace it. You may leave out the OR REPLACE if you don't want this to happen. The DECLARE section is now implicitly the section between the procedure definition and the BEGIN statement – it is declared after the procedure declaration. The END statement now ends a named block rather than an unnamed block. Nothing else is required for a procedure to run. To execute the stored procedure simply do the following.

```

SQL>EXEC MYPROC1;
PL/SQL procedure successfully completed.

```

**Functions:**

A Function is a subprogram that computes a value. It differs from Procedure as function returns a value.

The Syntax is:-

```

Function Name [(Argument [,Argument...])
Return Datatype Is
[Local Declaration]
Begin
Executable Statements
[Exception]
END [name];

```

Where argument stands for the following:-

```

Var_name [in | out | in out] datatype
[{:|=|DEFAULT} VALUE]

```

A function has two parts SPECIFICATION and Body. The specification begins with the keywords FUNCTION and ends with the RETURN clause which specified the

## Data Base Management Systems

the keyword **END** with an optional name. **BODY** has three parts as that of the procedure.

The function below demonstrates the syntax of a PL/SQL function block, note that as with a procedure the **OR REPLACE** clause can be left out if you don't want the replacement of an existing function with the same name. Function definition vary from procedure definitions in that you must explicitly name a variable to return and you must return a value in the variable via the **RETURN** statement.

```
CREATE OR REPLACE FUNCTION MYFUNC1
RETURN NUMBER
IS salary NUMBER(5);
BEGIN
SELECT SAL INTO salary FROM EMP where empno = 7369;
RETURN (salary);
END MYFUNC1;
```

```
Functional Sal_Ok (Salary Real, Title Read)
Return Boolean Is
Min_Sal Real;
Max_Sal Real;
Begin
Select Losal, Hisal Into Min_Sal, Max_Sal
From Sals Where Job = Title;
Return (Salary >= Min_Sal) And
(Salary <= Max_Sal);
End Sal_Ok;
```

If the salary is out of range **sal\_ok** is set to false; otherwise, **sal\_ok** is set to true. A function is called as a part of an expression. The function **sal\_ok** is called as

```
IF SAL_OK(NEW_SAL, NEW_TITLE) THEN
.....
ELSE
.....
END IF;
```

Functions should not be called inside SQL statements.

A **RETURN** statement immediately completes the execution of subprogram and returns control to the caller. A subprogram can contain several return statement. Executing any one of them completes the program.

```
Function Balance (Acc_Id Integer) Return Real Is
    Acct_Bal Real;
Begin
    Select Bal Into Acct_Bal From Accts
    Where Acct No = Acct_id;
    Return Acct_Bal;
End Balance;
```

## Data Base Management Systems

### Introduction to Triggers:-

Triggers are a special PL/SQL construct similar to procedures. However, a procedure is executed explicitly from another block via a procedure call, while a trigger is executed implicitly whenever the triggering event happens.

ORACLE allows you to define procedures that are implicitly executed when an insert, update, delete statement is issued against the associated table. These procedures are called database triggers. Triggers can be defined only on tables and not on views. The triggers which we use in FORMS are different since they are fired only when a trigger point is executed with in a specific application in FORMS. The database trigger is executed against a table, no matter what user or application issues the statement (insert, update, delete). So the triggering event is either a INSERT, DELETE, or UPDATE command.

Database triggers can be used to:

- Audit data modifications
- Log events transparently
- Enforce complex business rules
- Derive column values automatically
- Implement complex security authorizations
- Maintain replicate tables

A trigger has three basic parts:-

- A triggering event or statement
- A trigger restriction
- A trigger action

A trigger event or statement is the SQL statement that causes a trigger to be fired. A trigger event can be an insert, update or delete statement for a specific table.

A trigger restriction specifies a BOOLEAN expression that must be run for the trigger to fire. The trigger action wont take place if it evaluates to false. For example, it will fire when the condition is true.

**Quantity\_On\_Hand<Reorder\_Level**

A trigger action is the procedure that contains the SQL statements and PL/SQL code to be executed when a triggering statement is issued and the trigger restriction evaluates to true.

Some important points to note:

- You can create only BEFORE and AFTER triggers for tables. (INSTEAD OF triggers are only available for views; typically they are used to implement view updates.)
- You may specify up to three triggering events using the keyword OR. Furthermore, UPDATE can be optionally followed by the keyword OF a list of



## Data Base Management Systems

attribute(s) in <table\_name>. If present, the OF clause defines the event to be only an update of the attribute(s) listed after OF. Here are some examples:

... INSERT ON R ...  
 ... INSERT OR DELETE OR UPDATE ON R ...  
 ...UPDATE OF A,B OR INSERT ON R...

- If FOR EACH ROW option is specified, the trigger is row-level; otherwise, the trigger is statement-level.
- For a row-level trigger, a trigger restriction can be specified in the WHEN clause, enclosed by parentheses. The trigger restriction is a SQL condition that must be satisfied in order for ORACLE to fire the trigger. This condition cannot contain subqueries. Without the WHEN clause, a trigger is fired by every triggering event.
- <trigger\_body> is a PL/SQL block, rather than sequence of SQL statements. Oracle has placed certain restrictions on what you do in <trigger\_body>, in order to avoid situations where one trigger performs an action that triggers a second trigger, which then triggers a third, and so on, which could potentially create an infinite loop.

The restrictions on <trigger\_body> include:

- You cannot modify the same relation whose modification is the event triggering the trigger.
- You cannot modify a relation connected to the triggering relation by another constraint such as a foreign-key constraint.

### Types of Triggers

There are two kinds of triggers namely

- DML triggers and
- Instead-of triggers

In the case of DML triggers you can write triggers for insert, update or delete operations on a database table. In the case of instead-of triggers you can define triggers even on views. This kind of triggers can be defined on views only.

The general syntax of triggers is given below:

```
Create [or replace] trigger trigger_name
{ Before | After | Instead of } triggering_event
referencing_clause
[when trigger_condition]
[for each row]
trigger_body;
```

In the above syntax trigger name refers to the name of the trigger, triggering event specifies the event that fires the trigger. The referencing clause is used to refer to the data in the row currently being modified, trigger condition is a valid condition in the when clause and trigger\_body is the main code of the trigger.

## Data Base Management Systems

There are two levels of triggers namely **ROW** level and **STATEMENT** level. A row trigger is fired for each time the table is affected by the triggering statement. A statement type trigger is fired once on behalf of the triggering statement regardless of the number of rows in the table affected.

### TRIGGER TIMING

**BEFORE** triggers execute the trigger action before the triggering statement. **AFTER** triggers execute the action after the triggering statement is executed. So with these combinations we can create four types as follows:

- **BEFORE** statement trigger
- **BEFORE** row trigger
- **AFTER** statement trigger
- **After** row trigger