Database Systems: A Practical Approach to Design, Implementation, and Management

Sixth Edition

Chapter 1

Introduction to Databases

Pearson

Learning Objectives (1 of 2)

1.1 Some common uses of database systems.

1.2 Characteristics of file-based systems.

1.3 Problems with file-based approach.

1.4 Meaning of the term database.

1.5 Meaning of the term Database Management System (DBMS).

Learning Objectives (2 of 2)

1.6 Typical functions of a DBMS.

1.7 Major components of the DBMS environment.

1.8 Personnel involved in the DBMS environment.

1.9 History of the development of DBMSs.

1.10 Advantages and disadvantages of DBMSs.

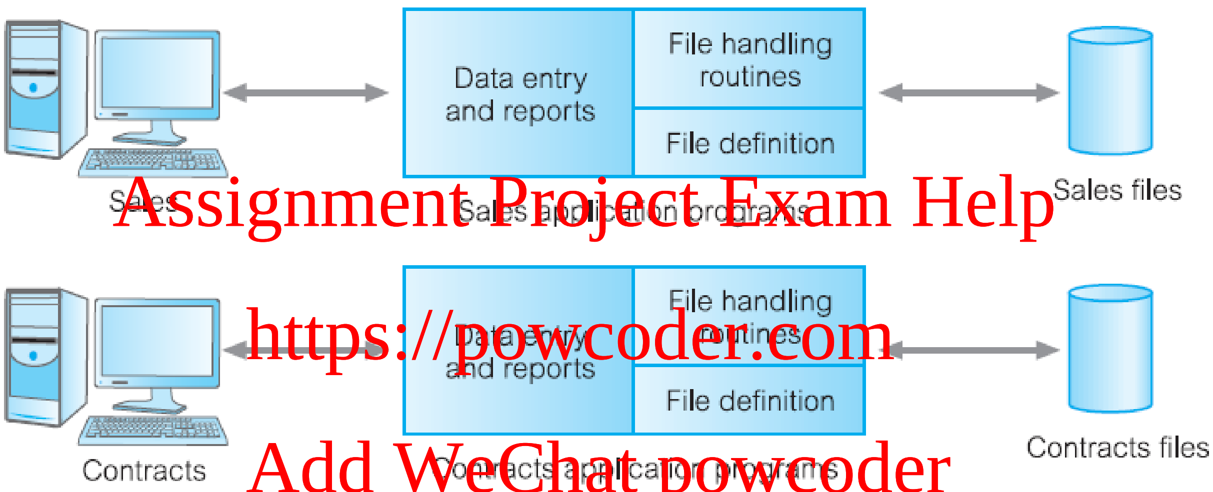
Examples of Database Applications

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

File-Based Systems

- Collection of application programs that perform services for the end users (e.g. reports).
- Each program defines and manages its own data.

File-Based Processing



Sales Files

- **PropertyForRent** (propertyNo, street, city, postcode, type, rooms, rent, and ownerNo)
- **PrivateOwner** (ownerNo, fName, lName, address, telNo)
- **Client** (clientNo, fName, lName, address, telNo, prefType, maxRent)

Contracts files

- **Lease** (leaseNo, propertyNo, clientNo, rent, paymentMethod, deposit, paid, rentStart, rentFinish, duration)
- **PropertyForRent** (propertyNo, street, city, postcode, rent)
- **Client** (clientNo, fName, lName, address, telNo)

Limitations of File-Based Approach (1 of 2)

- Separation and isolation of data
  - Each program maintains its own set of data.
  - Users of one program may be unaware of potentially useful data held by other programs.

- Duplication of data
  - Same data is held by different programs.
  - Wasted space and potentially different values and/or different formats for the same item.

Limitations of File-Based Approach (2 of 2)

- Data dependence
  - File structure is defined in the program code.

- Incompatible file formats
  - Programs are written in different languages, and so cannot easily access each other's files.

- Fixed Queries/Proliferation of application programs
  - Programs are written to satisfy particular functions.
  - Any new requirement needs a new program.

Database Approach (1 of 3)

- Arose because:
  - Definition of data was embedded in application programs, rather than being stored separately and independently.
  - No control over access and manipulation of data beyond that imposed by application programs.
- Result:
  - the database and Database Management System (DBMS).

Pearson

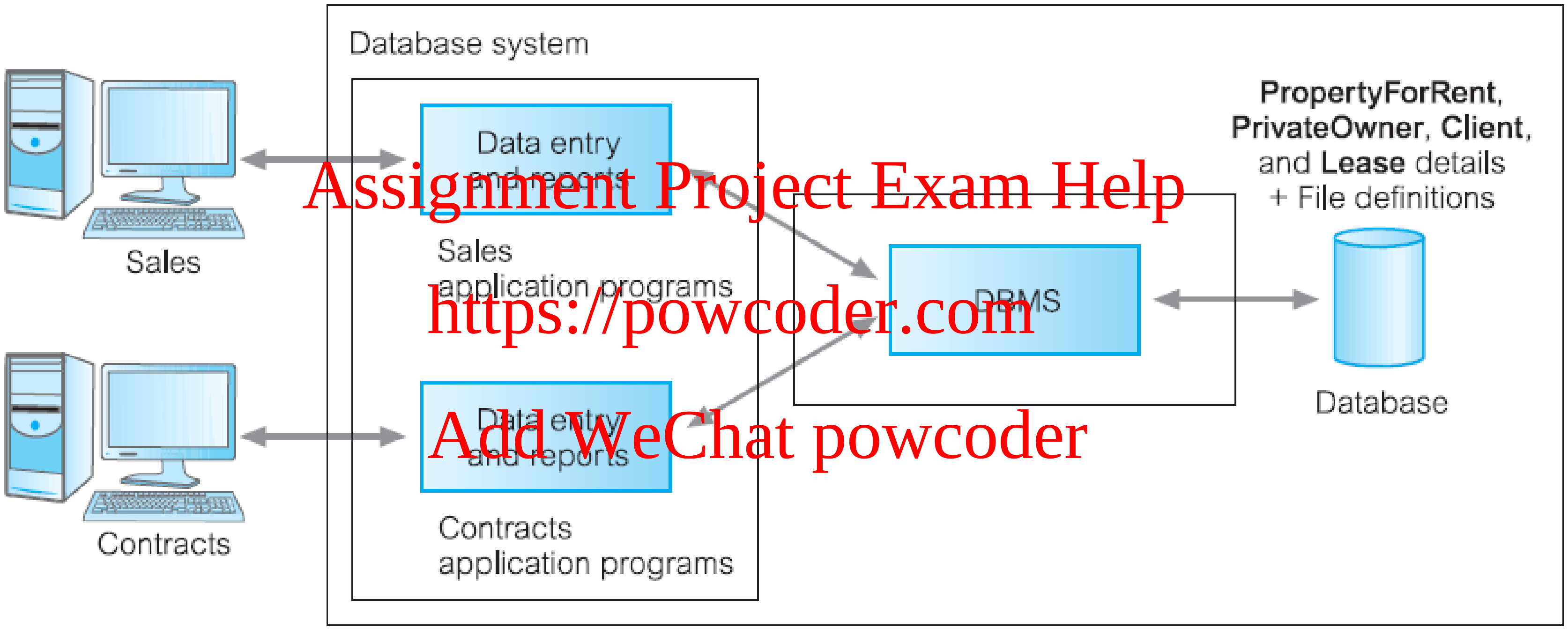
Database

- Shared collection of logically related data (and a description of this data), designed to meet the information needs of an organization.
- System catalog (metadata) provides description of data to enable program–data independence.
- Logically related data comprises entities, attributes, and relationships of an organization's information.

Database Management System (DBMS) (1 of 2)

- A software system that enables users to define, create, maintain, and control access to the database.
- (Database) application program: a computer program that interacts with database by issuing an appropriate request (SQL statement) to the DBMS.

Database Management System (DBMS) (2 of 2)



PropertyForRent (propertyNo, street, city, postcode, type, rooms, rent, ownerNo)
PrivateOwner (ownerNo, fname, lName, address, telNo)
Client (clientNo, fName, lName, address, telNo, prefType, maxRent)
Lease (leaseNo, propertyNo, clientNo, paymentMethod, deposit, paid, rentStart, rentFinish)

Database Approach (2 of 3)

- Data definition language (DDL).
  - Permits specification of data types, structures and any data constraints.
  - All specifications are stored in the database.
- Data manipulation language (DML).
  - General enquiry facility (query language) of the data.

Database Approach (3 of 3)

- Controlled access to database may include:
  - a security system
  - an integrity system
  - a concurrency control system
  - a recovery control system
  - a user-accessible catalog.

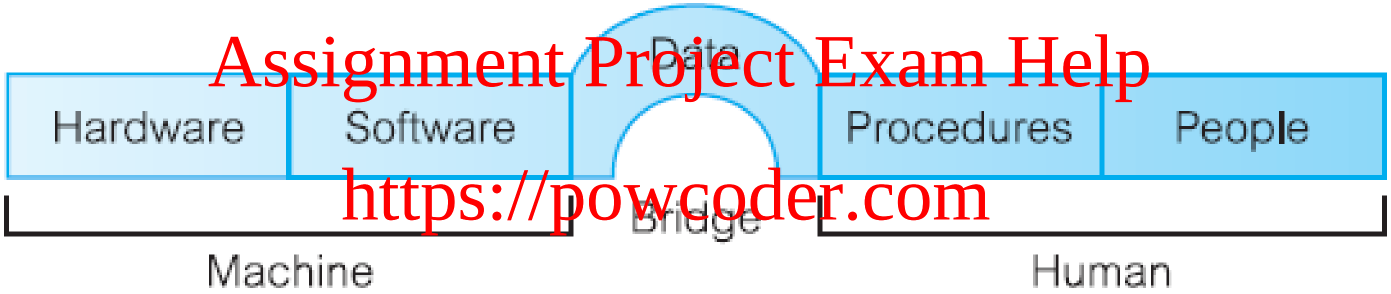
Views

- Allows each user to have his or her own view of the database.
- A view is essentially some subset of the database.

Views - Benefits

- Reduce complexity
- Provide a level of security
- Provide a mechanism to customize the appearance of the database
- Present a consistent, unchanging picture of the structure of the database, even if the underlying database is changed

Components of DBMS Environment (1 of 3)



Components of DBMS Environment (2 of 3)

- Hardware
  - Can range from a PC to a network of computers.
- Software
  - DBMS, operating system, network software (if necessary) and also the application programs.
- Data
  - Used by the organization and a description of this data called the schema.

Components of DBMS Environment (3 of 3)

- Procedures
  - Instructions and rules that should be applied to the design and use of the database and DBMS.
- People

Roles in the Database Environment

- Data Administrator (DA)
- Database Administrator (DBA)
- Database Designers (Logical and Physical)
- Application Programmers
- End Users (naive and sophisticated)

History of Database Systems

- First-generation
  - Hierarchical and Network

- Second generation
  - Relational

- Third generation

  - Object-Relational
  - Object-Oriented

Advantages of DBMSs (1 of 2)

- Control of data redundancy
- Data consistency
- More information from the same amount of data
- Sharing of data
- Improved data integrity
- Improved security
- Enforcement of standards
- Economy of scale

Advantages of DBMSs (2 of 2)

- Balance conflicting requirements
- Improved data accessibility and responsiveness

- Increased productivity

- Improved maintenance through data independence

- Increased concurrency
- Improved backup and recovery services

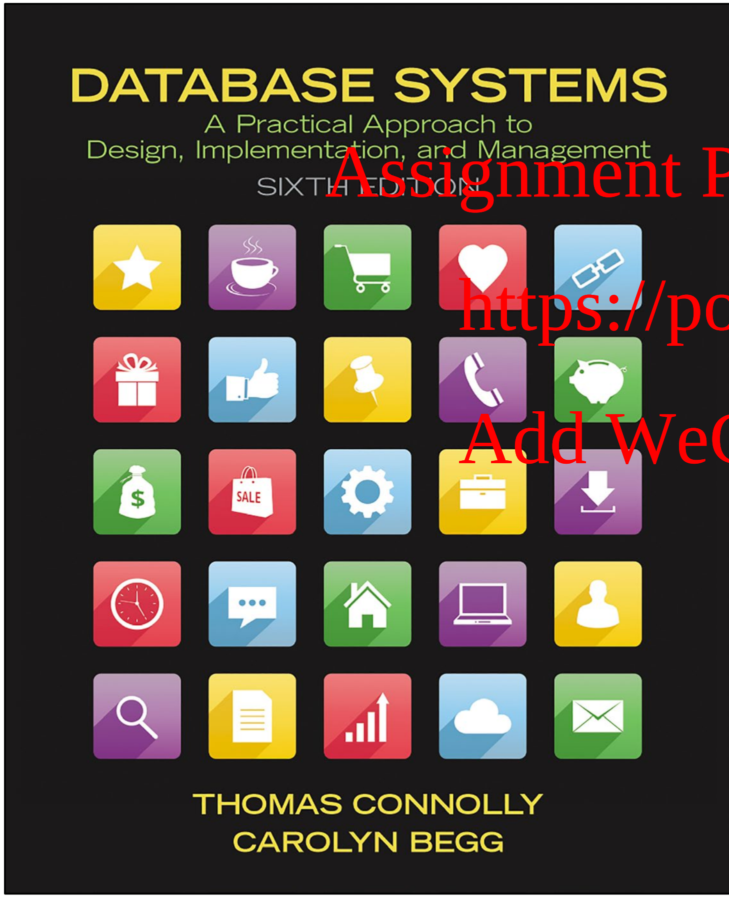
Disadvantages of DBMSs

- Complexity
- Size
- Cost of DBMS
- Additional hardware costs
- Cost of conversion
- Performance
- Higher impact of a failure

Copyright

Database Systems: A Practical Approach to Design, Implementation, and Management

Sixth Edition



Chapter 2

Database Environment

Learning Objectives (1 of 2)

2.1 Purpose of three-level database architecture.

2.2 Contents of external, conceptual, and internal levels.

2.3 Purpose of external/conceptual and conceptual/internal mappings.

2.4 Meaning of logical and physical data independence.

2.5 Distinction between DDL and DML.

2.6 A classification of data models.

Learning Objectives (2 of 2)

2.7 Purpose/importance of conceptual modeling.

2.8 Typical functions and services a DBMS should provide.

2.9 Function and importance of system catalog.

2.10 Software components of a DBMS.

2.11 Meaning of client–server architecture and advantages of this type of architecture for a DBMS.

2.12 Function and uses of Transaction Processing Monitors.

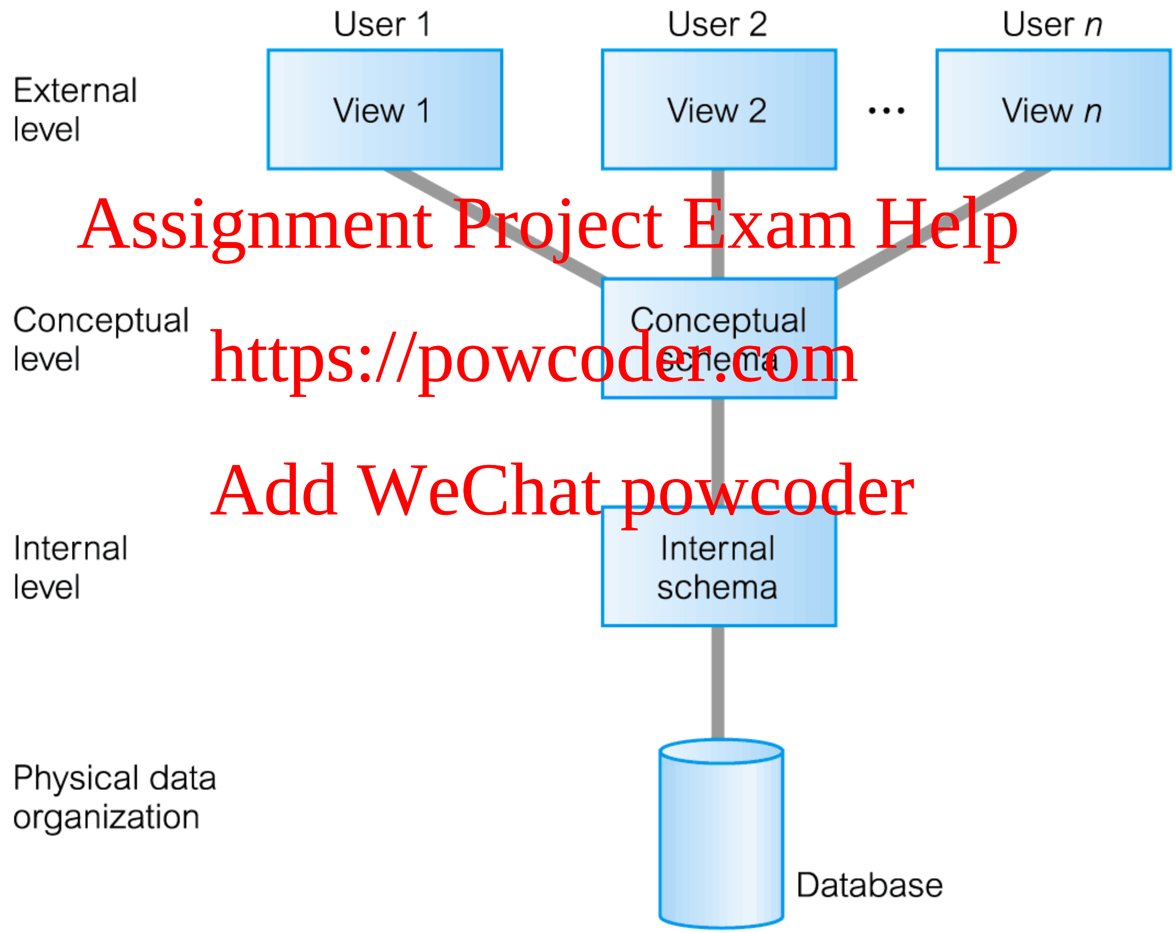
Objectives of Three-Level Architecture (1 of 2)

- All users should be able to access same data.
- A user's view is immune to changes made in other views.
- Users should not need to know physical database storage details.

Objectives of Three-Level Architecture (2 of 2)

- DBA should be able to change database storage structures without affecting the users' views.
- Internal structure of database should be unaffected by changes to physical aspects of storage.
- DBA should be able to change conceptual structure of database without affecting all users.

ANSI-SPARC Three-Level Architecture (1 of 3)



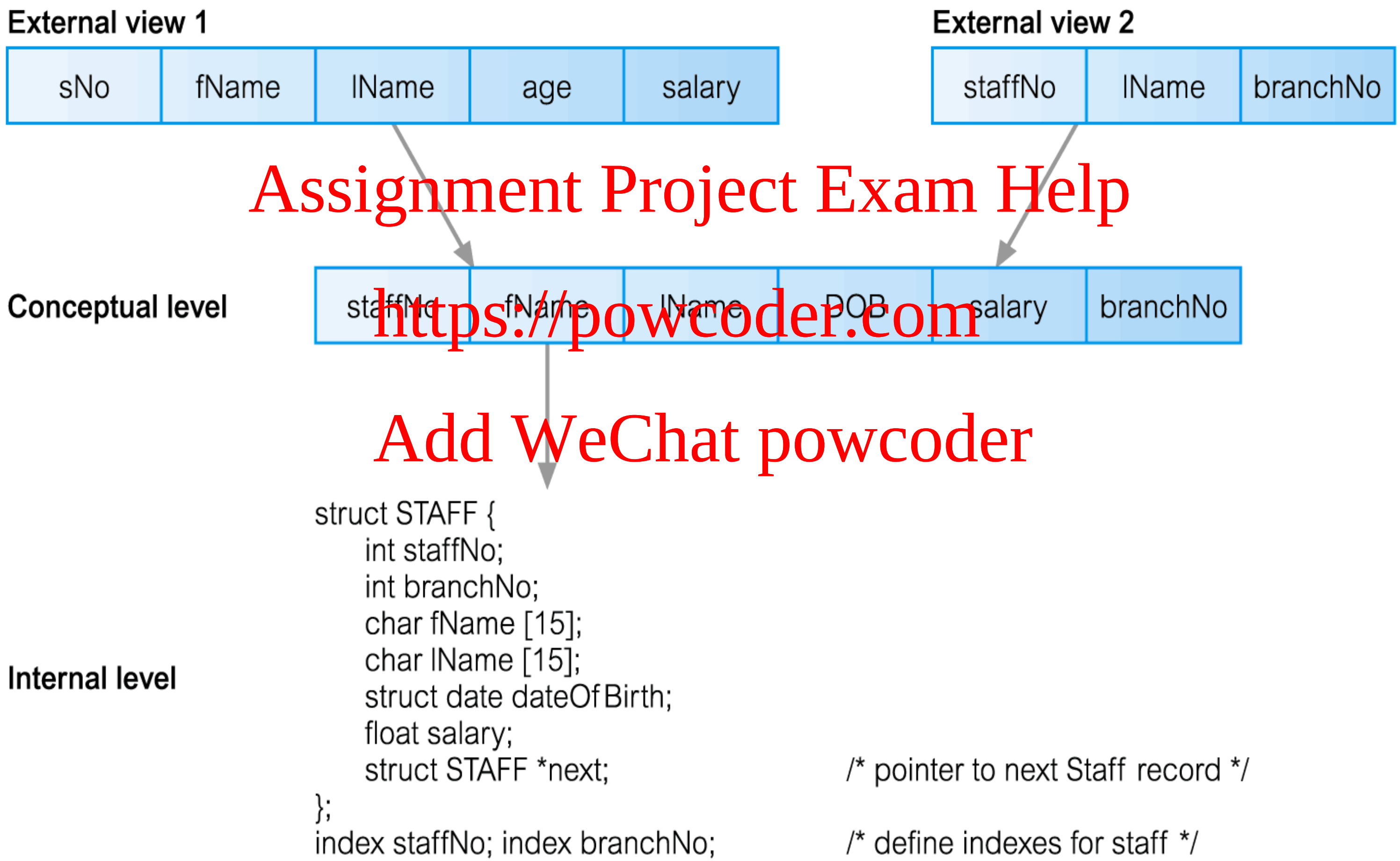
ANSI-SPARC Three-Level Architecture (2 of 3)

- External Level
  - Users' view of the database.
  - Describes that part of database that is relevant to a particular user.
- Conceptual Level
  - Community view of the database.
  - Describes what data is stored in database and relationships among the data.

ANSI-SPARC Three-Level Architecture (3 of 3)

- Internal Level
  - Physical representation of the database on the computer.
  - Describes how the data is stored in the database.

Differences Between Three Levels of ANSI-SPARC Architecture



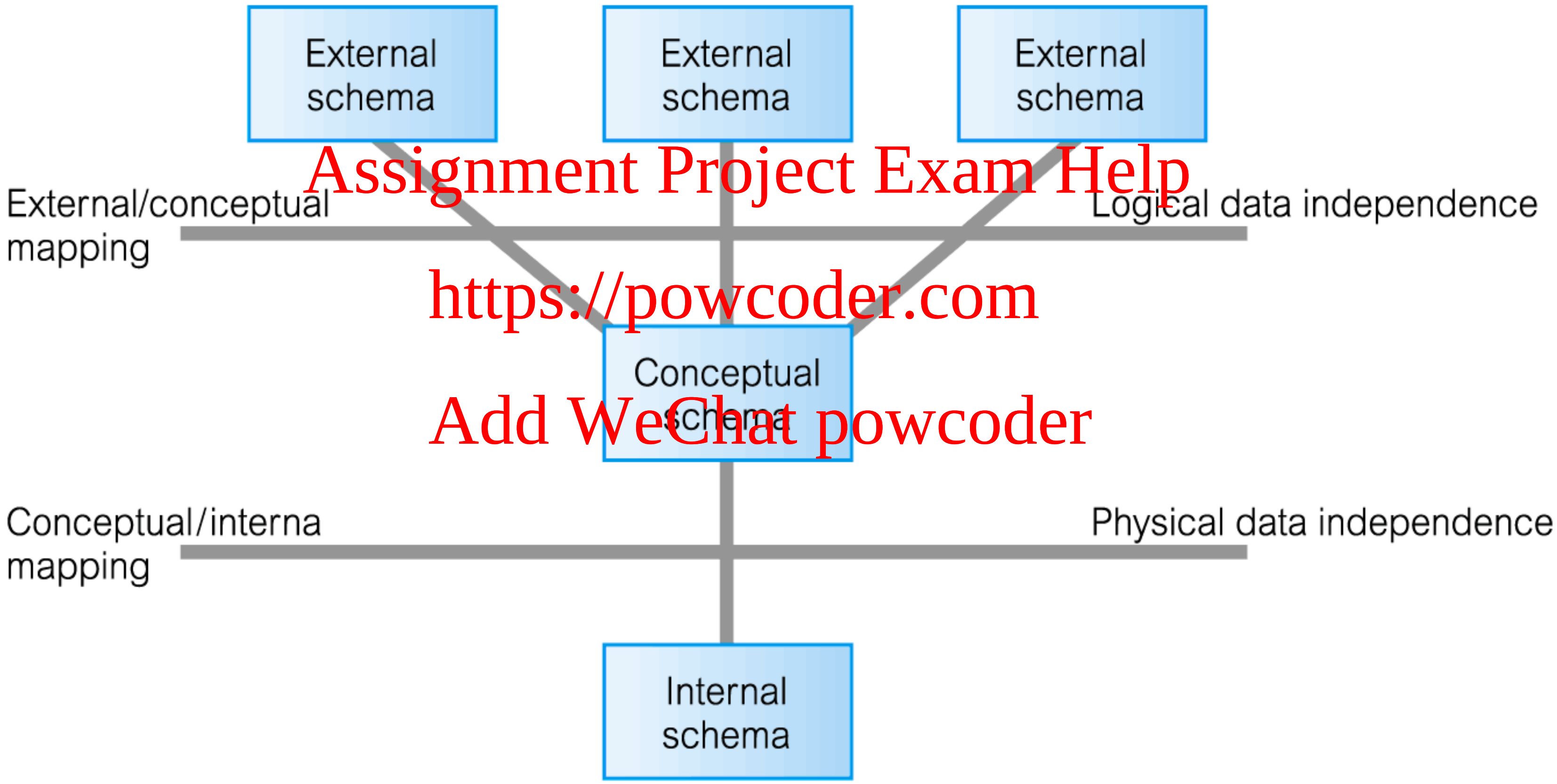
Data Independence (1 of 2)

- Logical Data Independence
  - Refers to immunity of external schemas to changes in conceptual schema.
  - Conceptual schema changes (e.g. addition/removal of entities).
  - Should not require changes to external schema or rewrites of application programs.

Data Independence (2 of 2)

- Physical Data Independence
  - Refers to immunity of conceptual schema to changes in the internal schema.
  - Internal schema changes (e.g. using different file organizations, storage structures/devices).
  - Should not require change to conceptual or external schemas.

Data Independence and the ANSI-SPARC Three-Level Architecture



Database Languages (1 of 2)

- Data Definition Language (DDL)
  - Allows the DBA or user to describe and name entities, attributes, and relationships required for the application
  - plus any associated integrit and security constraints.

Database Languages (2 of 2)

- Data Manipulation Language (DML)
  - Provides basic data manipulation operations on data held in the database.

- Procedural DML

  - allows user to tell system exactly how to manipulate data.

- Non-Procedural DML
  - allows user to state what data is needed rather than how it is to be retrieved.
- Fourth Generation Languages (4GLs)

Data Model (1 of 2)

- Integrated collection of concepts for describing data, relationships between data, and constraints on the data in an organization.
- Data Model comprises:
  - a structural part;
  - a manipulative part;
  - possibly a set of integrity rules.

Data Model (2 of 2)

- Purpose
  - To represent data in an understandable way.
- Categories of data models include:
  - Object-based
  - Record-based
  - Physical.

Data Models

- Object-Based Data Models
  - Entity-Relationship
  - Semantic
  - Functional
  - Object-Oriented.
- Record-Based Data Models
  - Relational Data Model
  - Network Data Model
  - Hierarchical Data Model.
- Physical Data Models

Relational Data Model

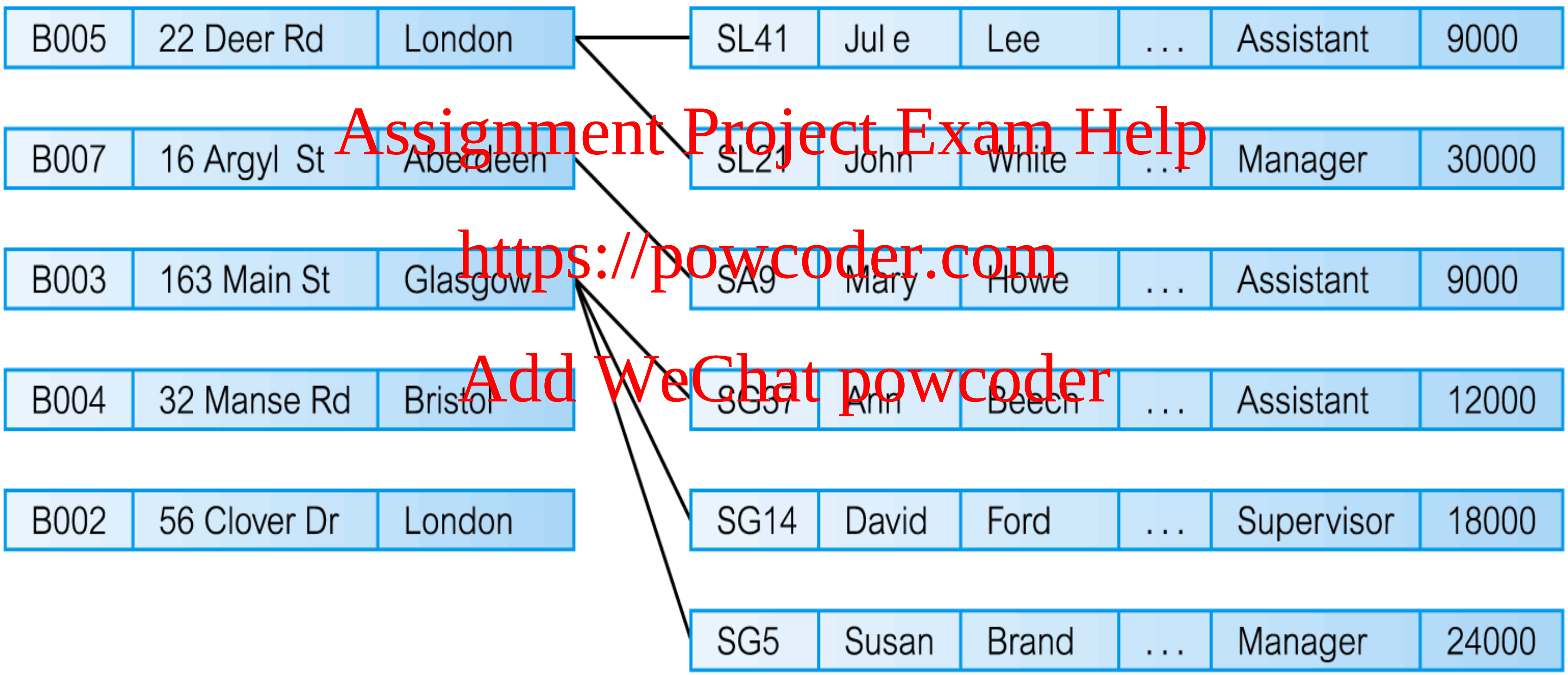
Branch

branchNo	street	City	postCode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

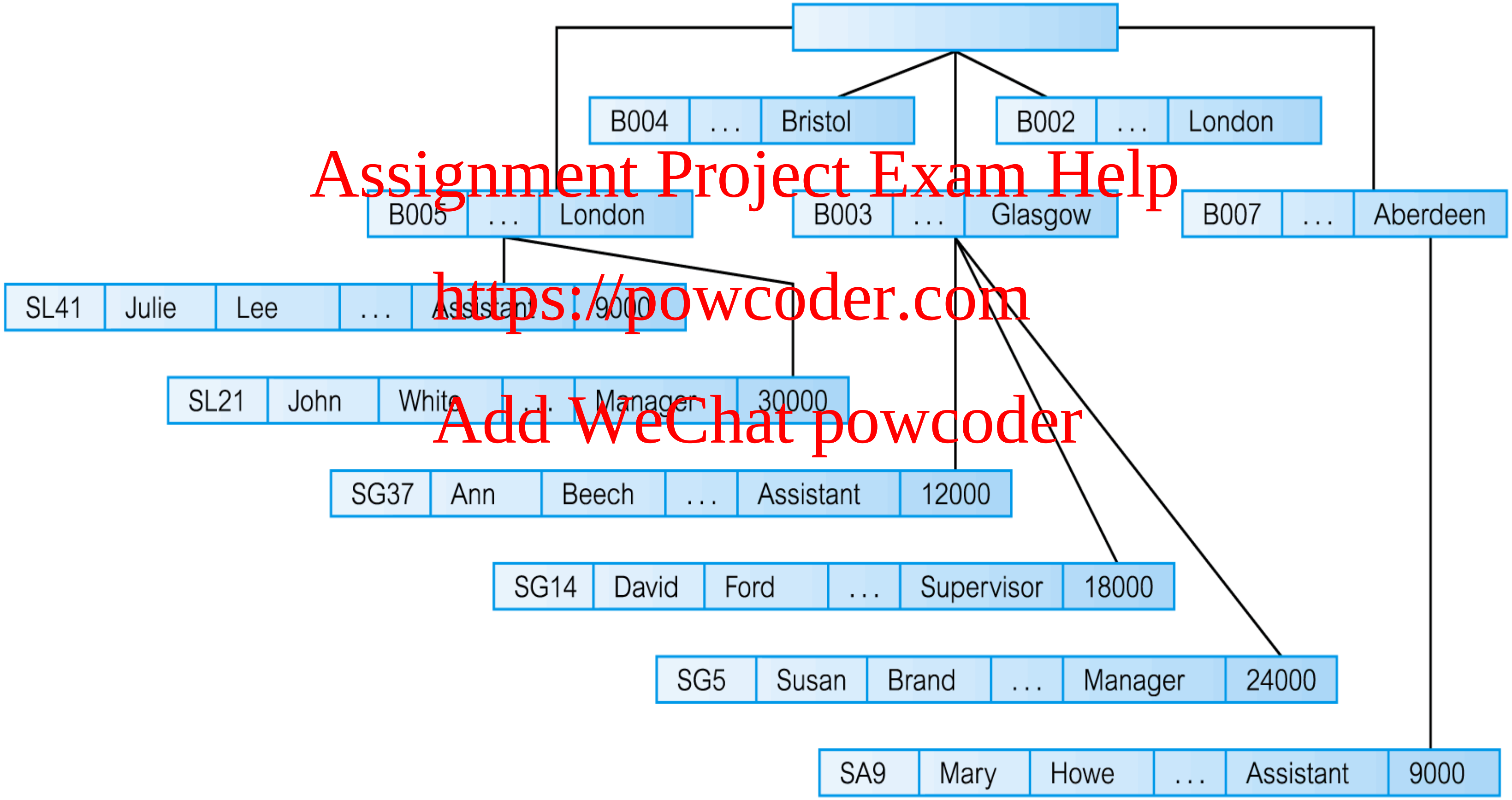
Staff

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
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Network Data Model



Hierarchical Data Model



Conceptual Modeling

- Conceptual schema is the core of a system supporting all user views.
- Should be complete and accurate representation of an organization's data requirements.
- Conceptual modeling is process of developing a model of information use that is independent of implementation details.
- Result is a conceptual data model.

Functions of a DBMS (1 of 2)

- Data Storage, Retrieval, and Update.
- A User-Accessible Catalog.
- Transaction Support.
- Concurrency Control Services.
- Recovery Services.

Functions of a DBMS (2 of 2)

- Authorization Services.
- Support for Data Communication.

- Integrity Services.

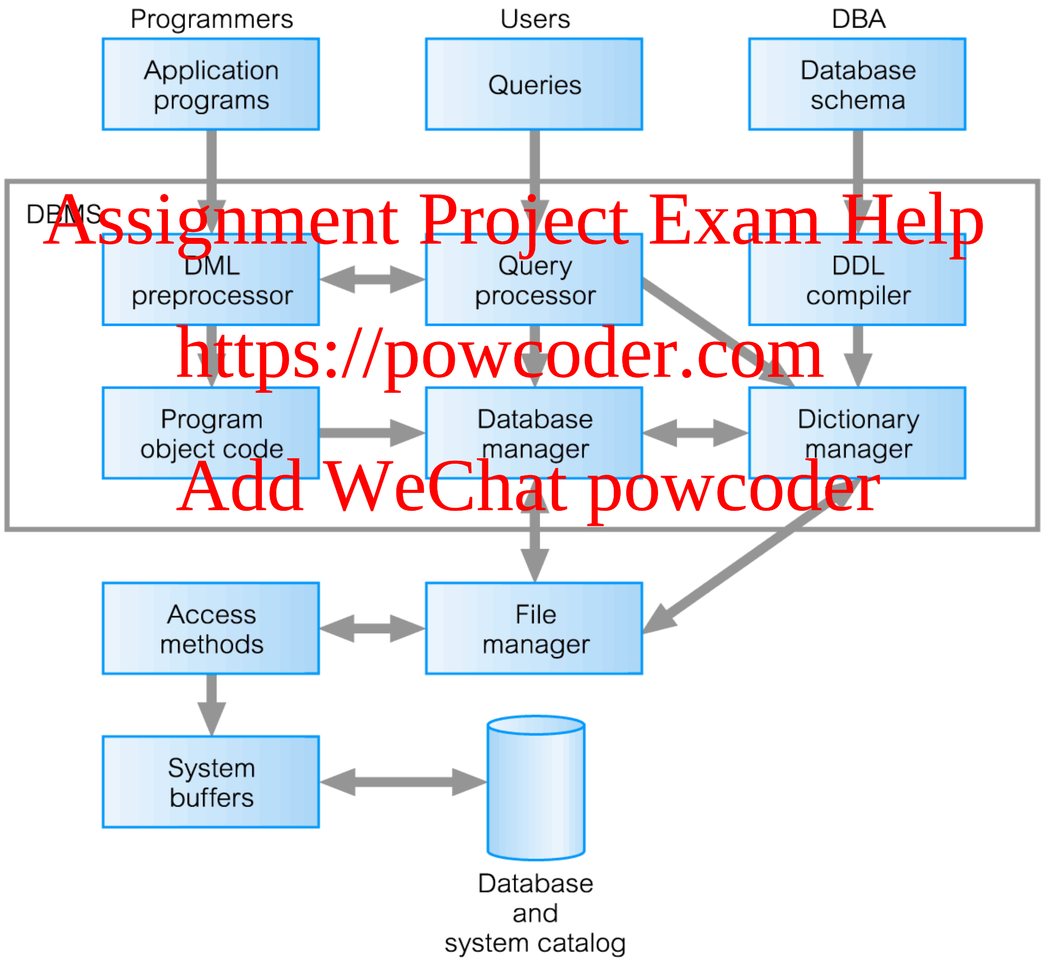
- Services to Promote Data Independence.

- Utility Services.

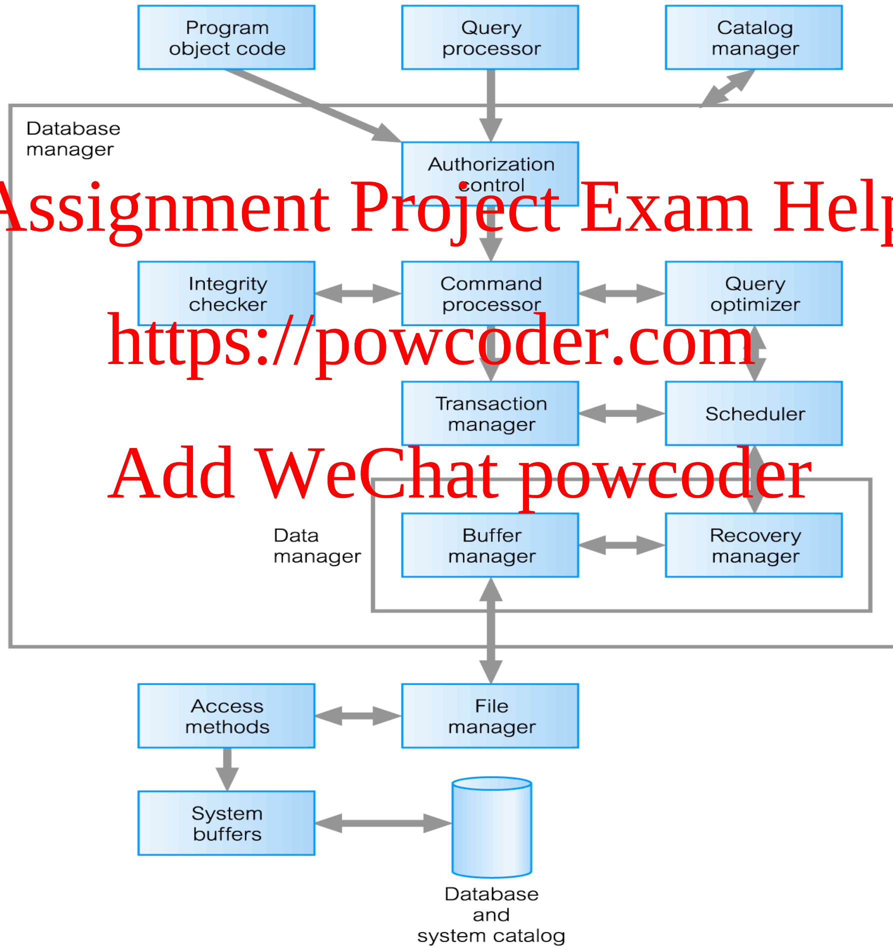
System Catalog

- Repository of information (metadata) describing the data in the database.
- One of the fundamental components of DBMS.
- Typically stores:
  - names, types, and sizes of data items;
  - constraints on the data;
  - names of authorized users;
  - data items accessible by a user and the type of access;
  - usage statistics.

Components of a DBMS



Components of Database Manager



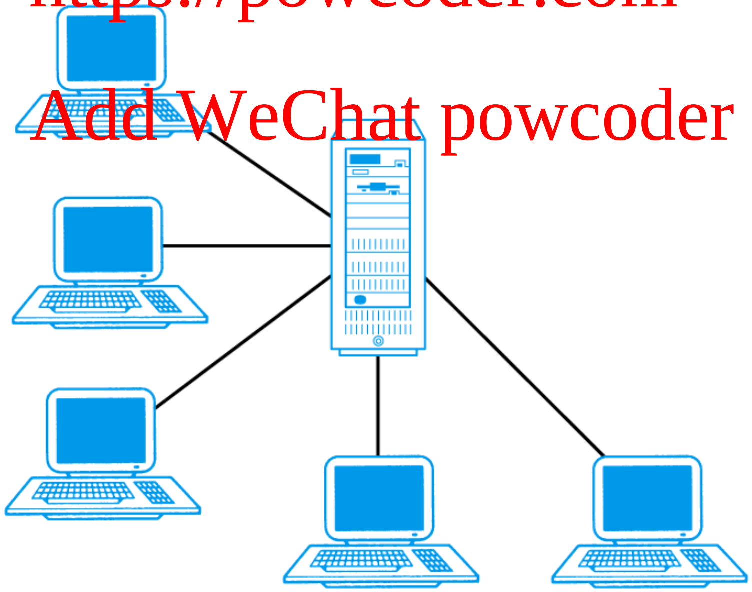
Multi-User DBMS Architectures

- Teleprocessing
- File-server
- Client-server

Teleprocessing

- Traditional architecture.
- Single mainframe with a number of terminals attached.

- Trend is now towards downsizing.



File-Server

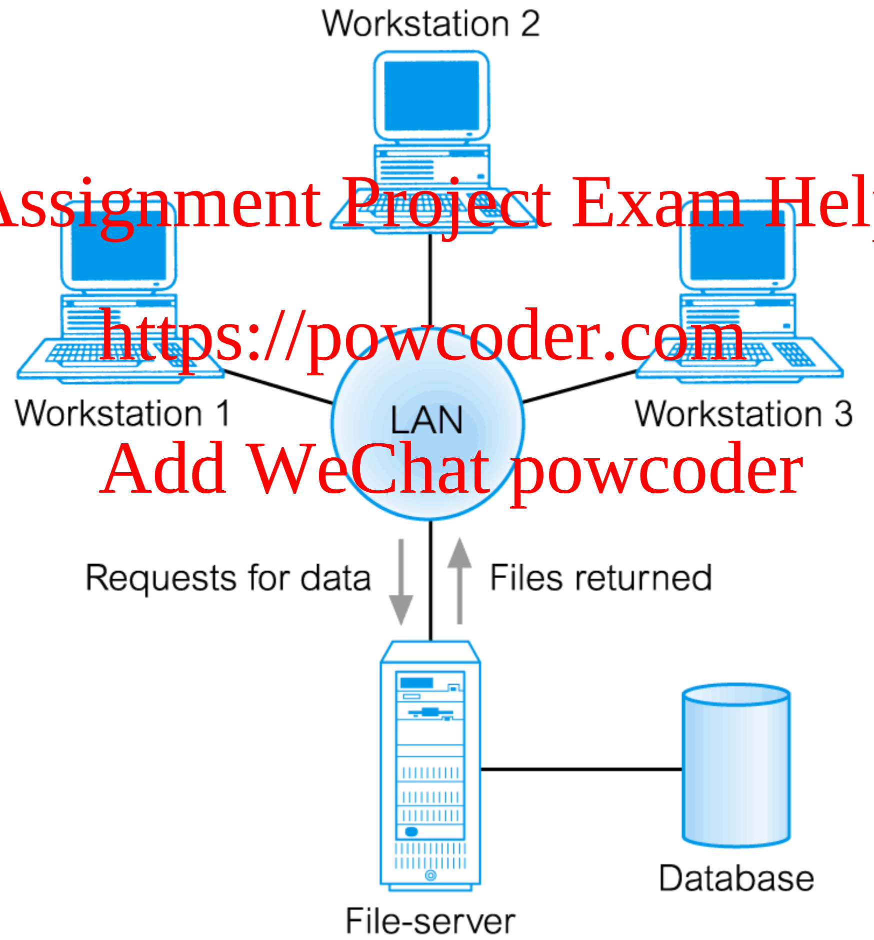
- File-server is connected to several workstations across a network.
- Database resides on file-server.

- DBMS and applications run on each workstation.

- Disadvantages include:

  - Significant network traffic.
  - Copy of DBMS on each workstation.
  - Concurrency, recovery and integrity control more complex.

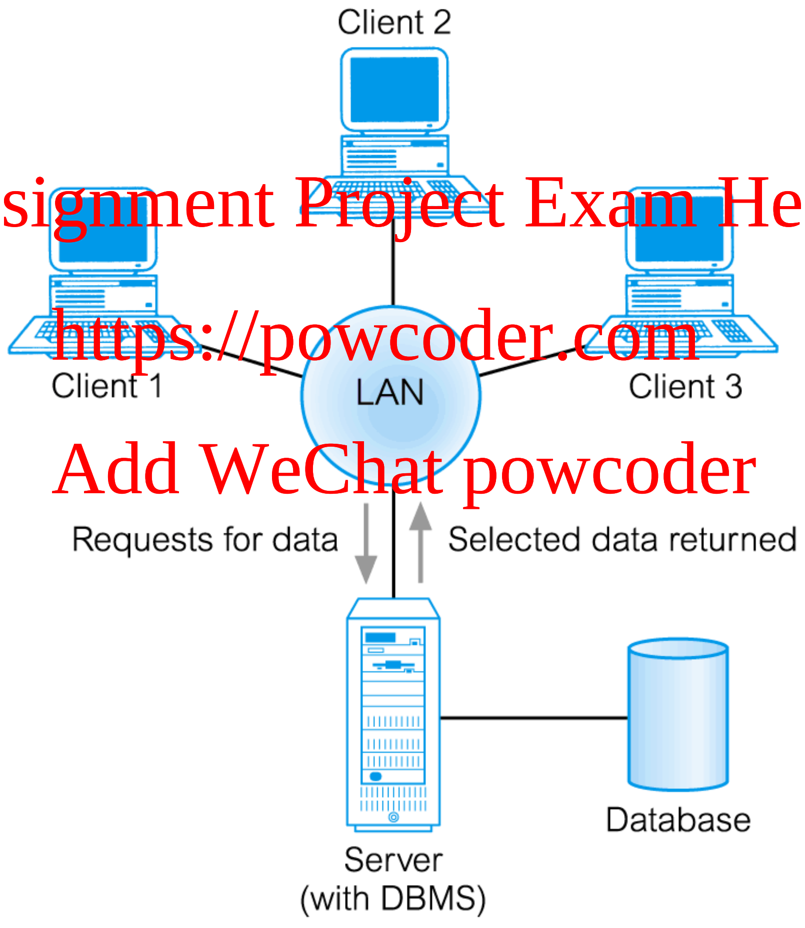
File-Server Architecture



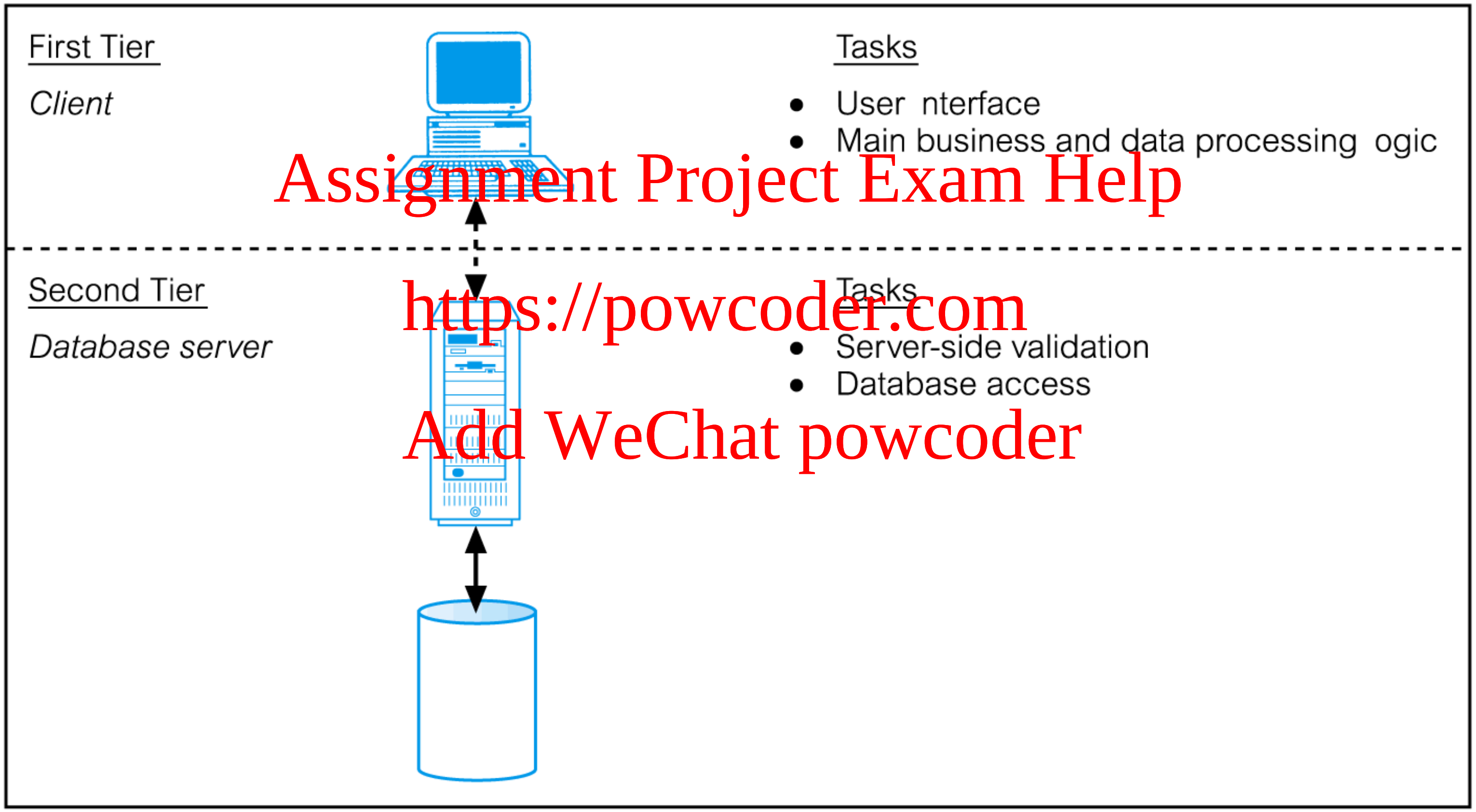
Traditional Two-Tier Client-Server (1 of 3)

- Client (tier 1) manages user interface and runs applications.
- Server (tier 2) holds database and DBMS.
- Advantages include:
  - wider access to existing databases;
  - increased performance;
  - possible reduction in hardware costs;
  - reduction in communication costs;
  - increased consistency.

Traditional Two-Tier Client-Server (2 of 3)



Traditional Two-Tier Client-Server (3 of 3)



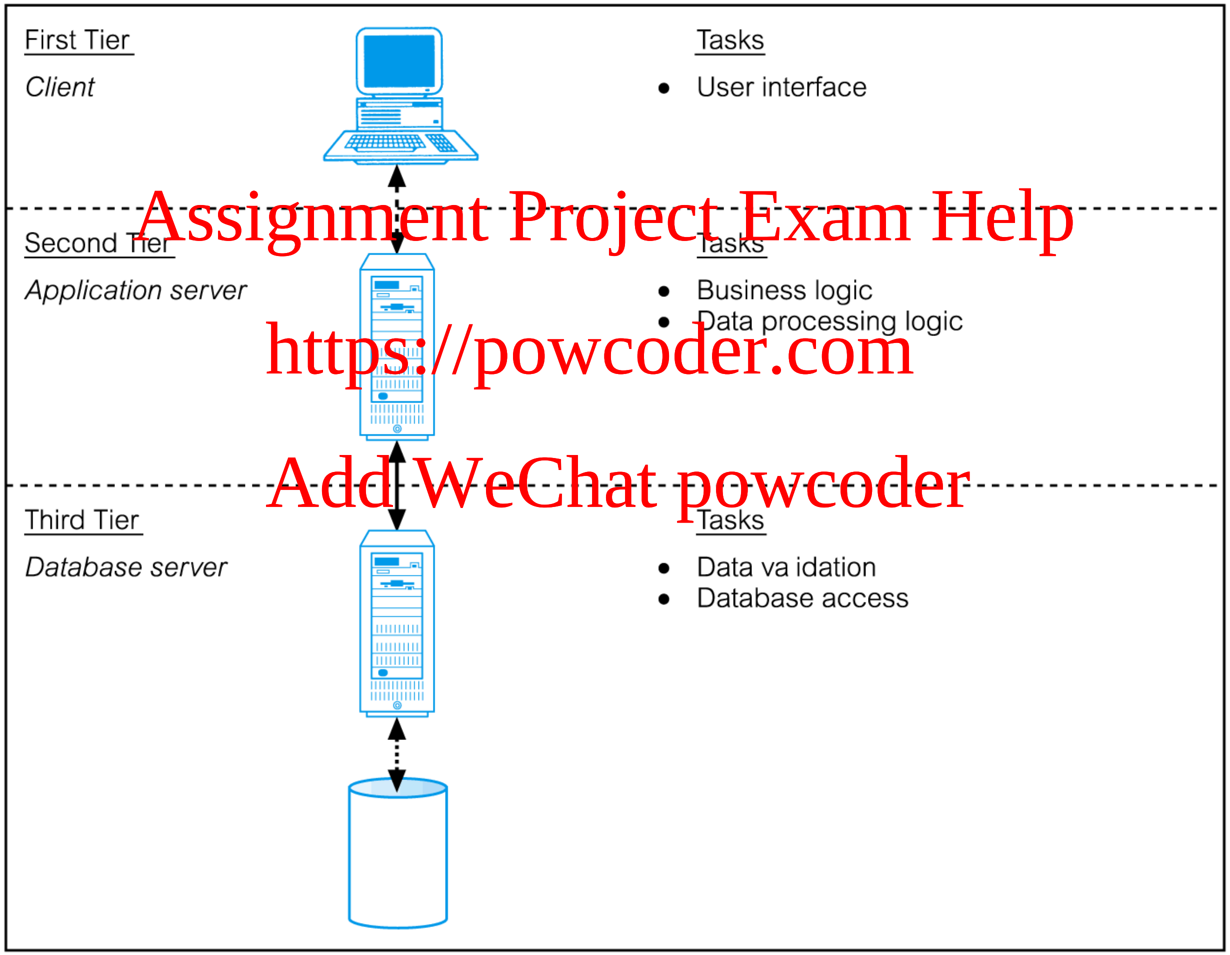
Three-Tier Client-Server (1 of 3)

- Client side presented two problems preventing true scalability:
  - ‘Fat’ client, requiring considerable resources on client’s computer to run effectively.
  - Significant client side administration overhead.
- By 1995, three layers proposed, each potentially running on a different platform.

Three-Tier Client-Server (2 of 3)

- Advantages:
  - ‘Thin’ client, requiring less expensive hardware.
  - Application maintenance centralized.
  - Easier to modify or replace one tier without affecting others.
  - Separating business logic from database functions makes it easier to implement load balancing.
  - Maps quite naturally to Web environment.

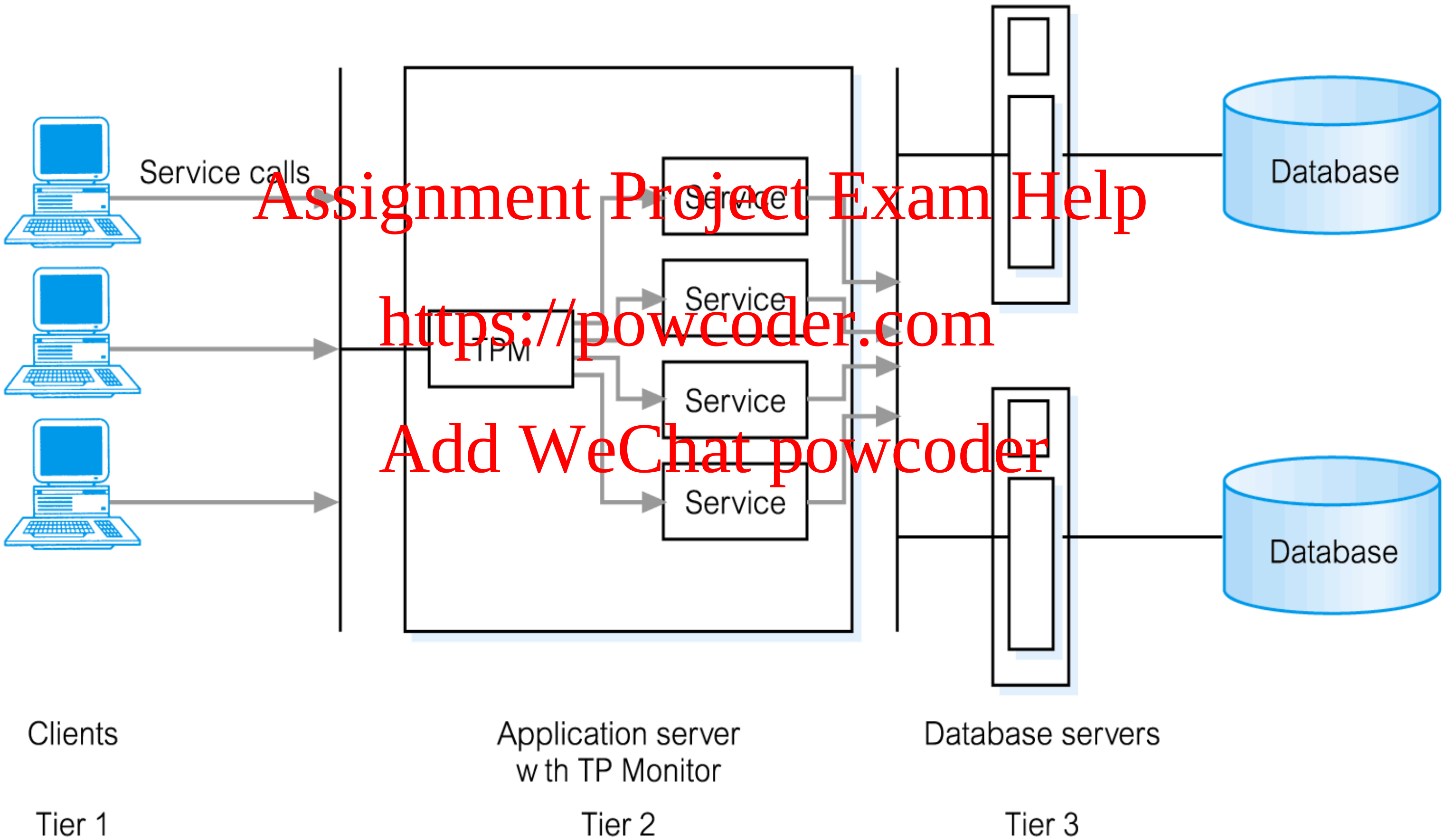
Three-Tier Client-Server (3 of 3)



Transaction Processing Monitors

- Program that controls data transfer between clients and servers in order to provide a consistent environment, particularly for Online Transaction Processing (OLTP).

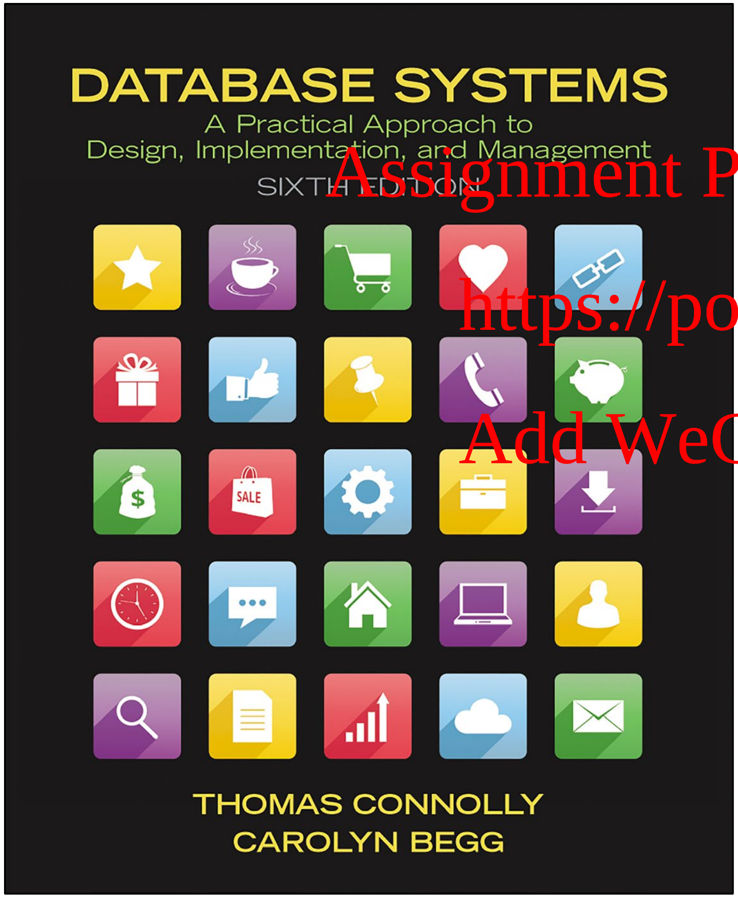
TPM as Middle Tier of 3-Tier Client-Server



Copyright

Database Systems: A Practical Approach to Design, Implementation, and Management

Sixth Edition



Chapter 3

Database Architectures and the Web Transparencies

Learning Objectives (1 of 2)

3.1 The meaning of the client–server architecture and the advantages of this type of architecture for a DBMS.

3.2 The difference between two-tier, three-tier and **n**-tier client–server architectures.

3.3 About cloud computing and data as a service (DaaS) and database as a service (DBaaS).

3.4 Software components of a DBMS.

Learning Objectives (2 of 2)

3.5 The purpose of a Web service and the technological standards used to develop a Web service.

3.6 The meaning of service-oriented architecture (SOA).

3.7 The difference between distributed DBMSs, and distributed processing.

3.8 The architecture of a data warehouse.

3.9 About cloud computing and cloud databases.

3.10 The software components of a DBMS.

Multi-User DBMS Architectures

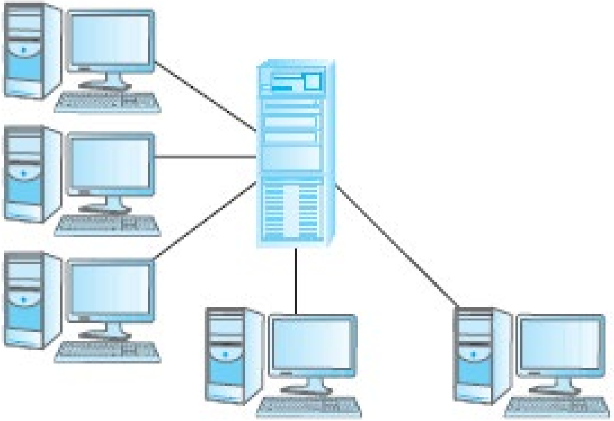
- The common architectures that are used to implement multi-user database management systems:
  - Teleprocessing
  - File-Server
  - Client-Server

File-Server

- File-server is connected to several workstations across a network.
- Database resides on file-server.
- DBMS and applications run on each workstation.
- Disadvantages include:
  - Significant network traffic.
  - Copy of DBMS on each workstation.
  - Concurrency, recovery and integrity control more complex.

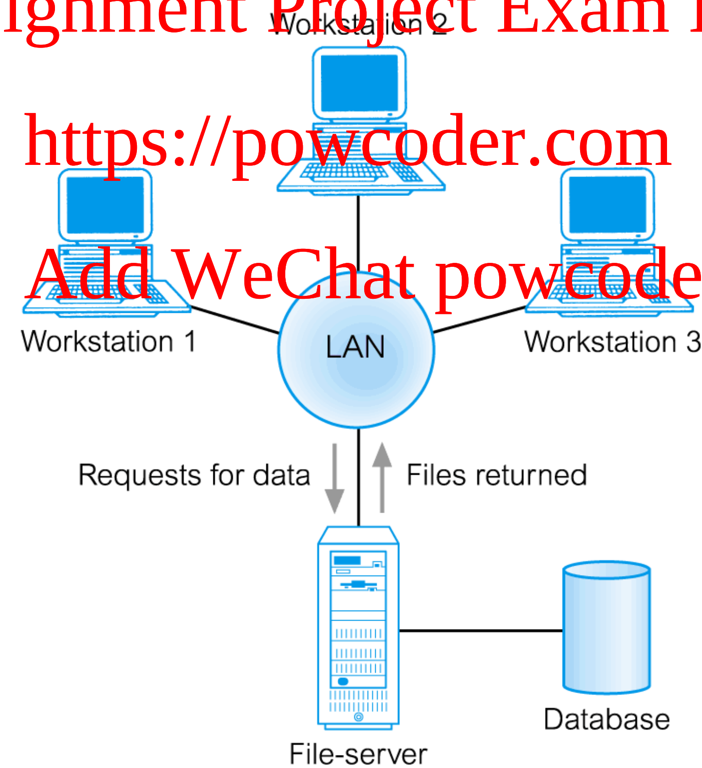
Teleprocessing

- One computer with a single CPU and a number of terminals.
- Processing performed within the same physical computer. User terminals are typically "dumb", incapable of functioning on their own, and cabled to the central computer.



File-Server Architecture

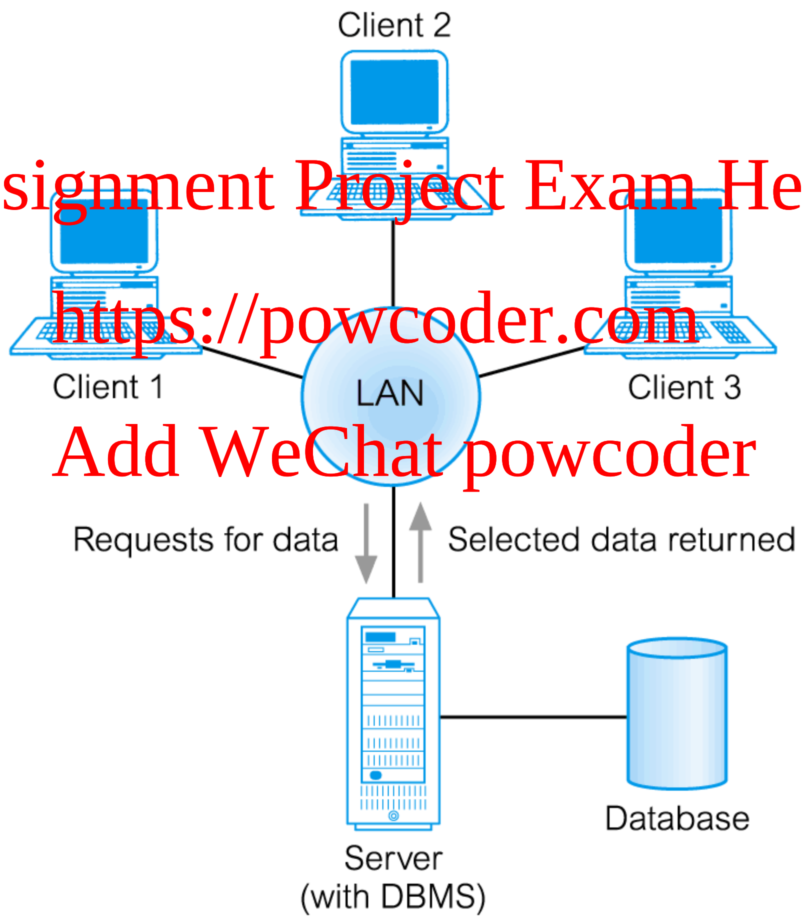
- In a file-server environment, the processing is distributed about the network, typically a local area network (LAN).



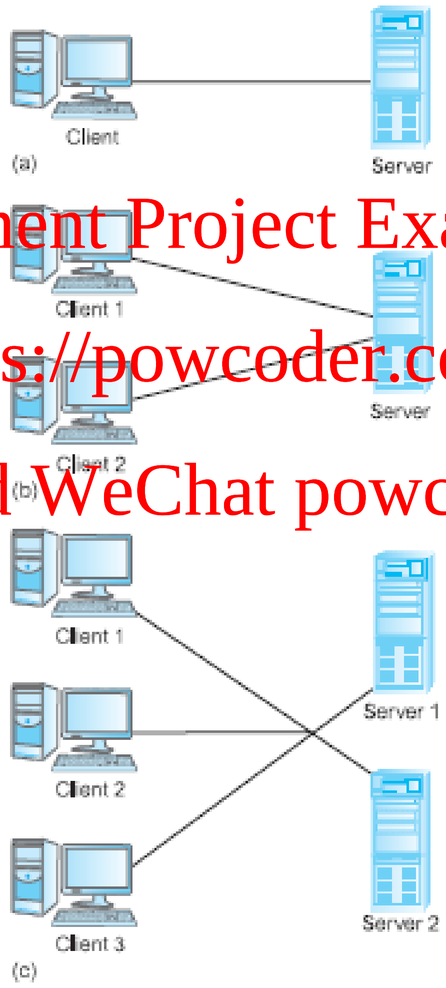
Traditional Two-Tier Client-Server (1 of 3)

- Client (tier 1) manages user interface and runs applications.
- Server (tier 2) holds database and DBMS.
- Advantages include:
  - wider access to existing databases;
  - increased performance;
  - possible reduction in hardware costs;
  - reduction in communication costs;
  - increased consistency.

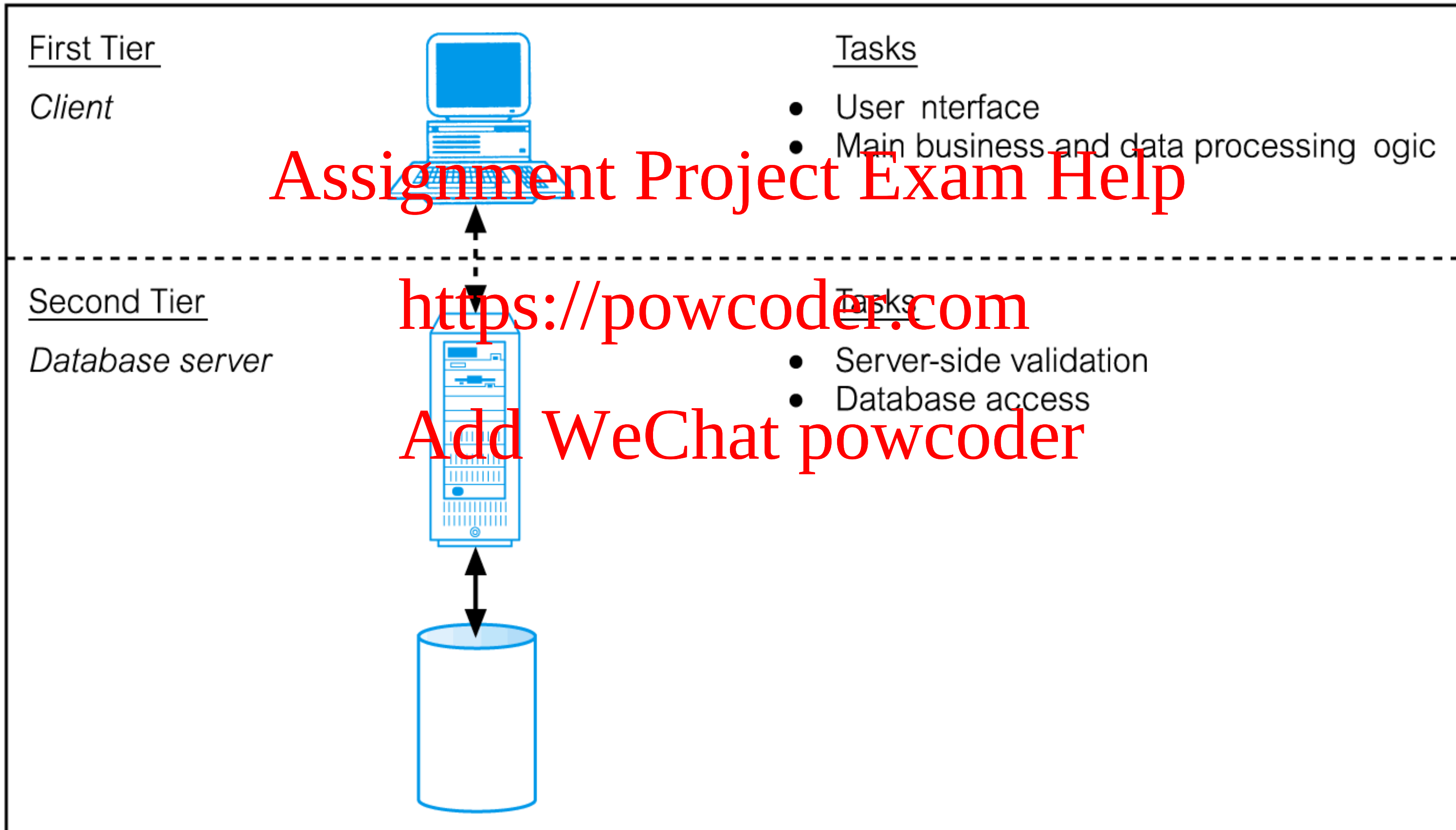
Traditional Two-Tier Client-Server (2 of 3)



Alternative Client-Server Topologies



Traditional Two-Tier Client-Server (3 of 3)



Summary of Client-Server Functions

Client	Server
Manages the user interface	Accepts and processes database requests from clients
Accepts and checks syntax of user input	Checks authorization
Processes application logic	Ensures integrity constraints not violated
Generates database requests and transmits to server	Performs query/update processing and transmits response to client
Passes response back to user	Maintains system catalog Provides concurrent database access Provides recovery control

Three-Tier Client-Server (1 of 3)

- The need for enterprise scalability challenged the traditional two-tier client–server model.

- Client side presented two problems preventing true scalability:

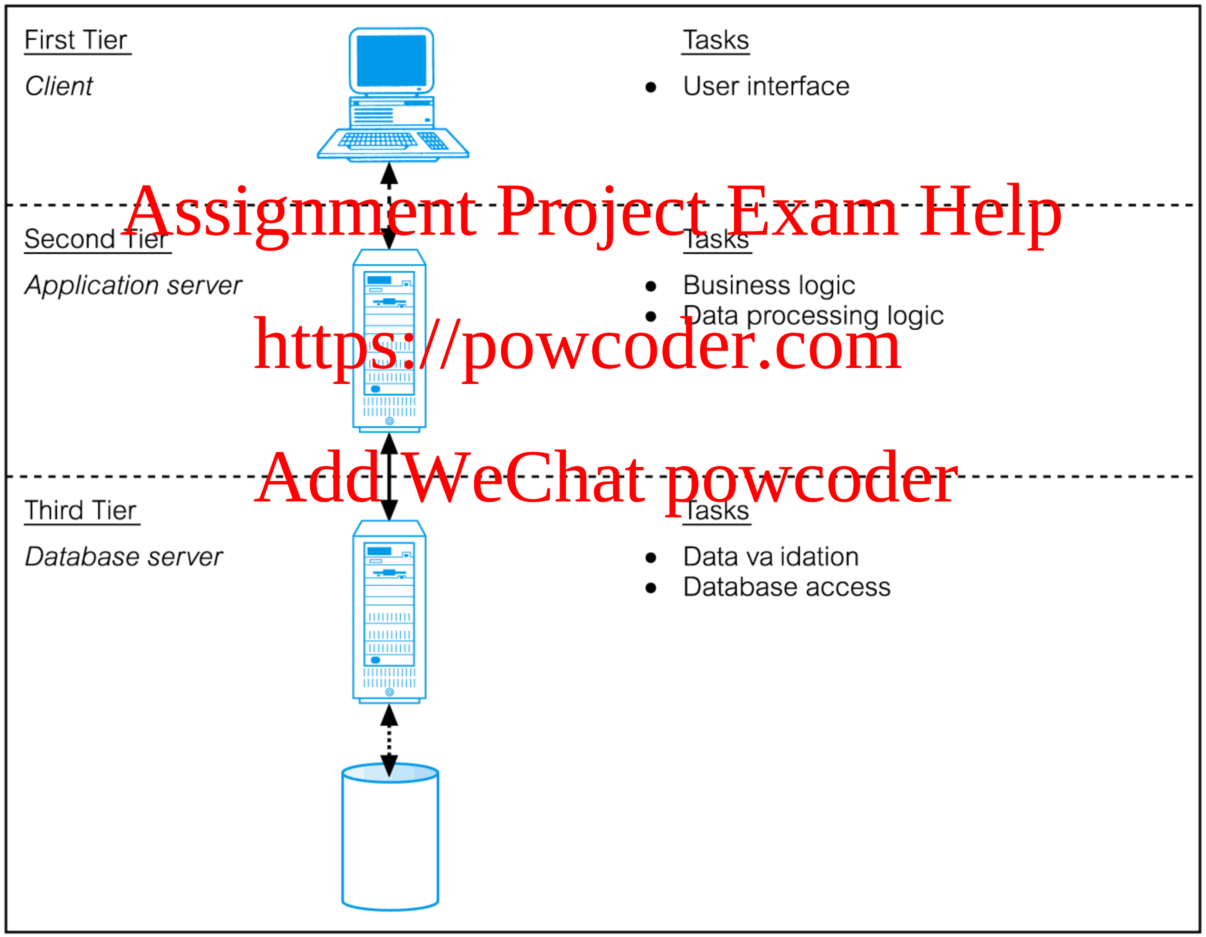
  - ‘Fat’ client, requiring considerable resources on client’s computer to run effectively.

  - Significant client side administration overhead.
- By 1995, three layers proposed, each potentially running on a different platform.

Three-Tier Client-Server (2 of 3)

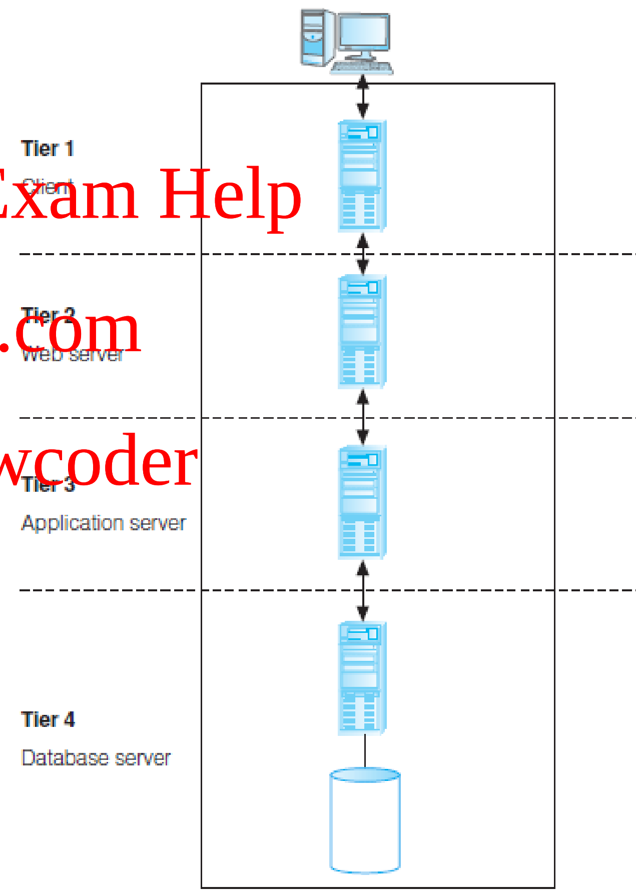
- Advantages:
  - ‘Thin’ client, requiring less expensive hardware.
  - Application maintenance centralized.
  - Easier to modify or replace one tier without affecting others.
  - Separating business logic from database functions makes it easier to implement load balancing.
  - Maps quite naturally to Web environment.

Three-Tier Client-Server (3 of 3)



n-Tier Client-Server (e.g. 4-Tier)

- The three-tier architecture can be expanded to **n** tiers, with additional tiers providing more flexibility and scalability.
- Applications servers host API to expose business logic and business processes for use by other applications.



Middleware

- Middleware is a generic term used to describe software that mediates with other software and allows for communication between disparate applications in a heterogeneous system.
- The need for middleware arises when distributed systems become too complex to manage efficiently without a common interface.

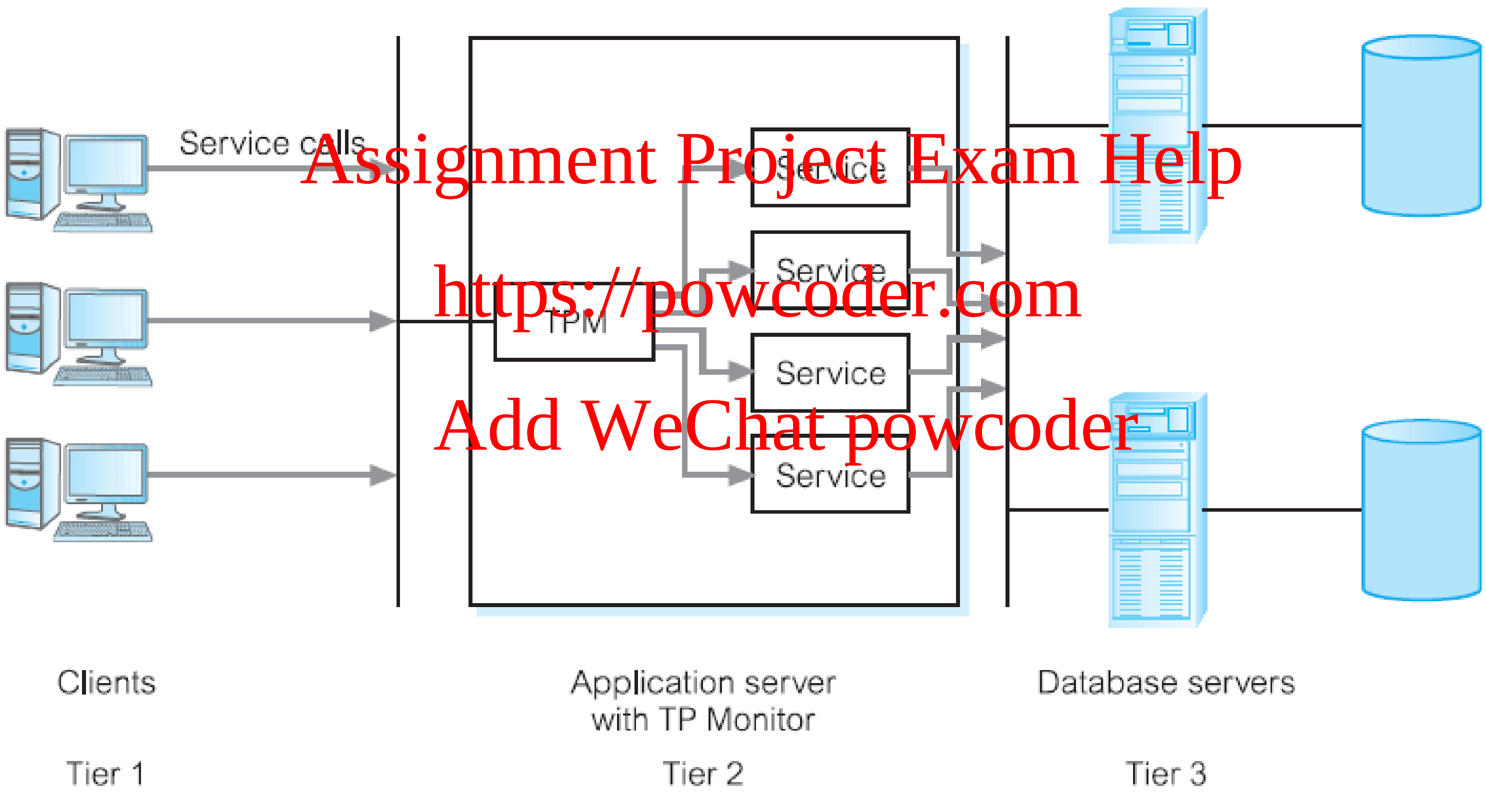
Cloud Computing (1 of 2)

- The National Institute of Standards and Technology (NIST) provided a definition.
- Defined as "A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Transaction Processing Monitors

- TP monitor is a program that controls data transfer between clients and servers in order to provide a consistent environment, particularly for online transaction processing (OLTP).

Transaction Processing Monitor as Middle Tier of 3-Tier Client-Server



Web Services and Service-Oriented Architectures (1 of 3)

- Web service is a software system designed to support interoperable machine-to-web service machine interaction over a network.

- Web services share business logic, data, and processes through a programmatic interface across a network.

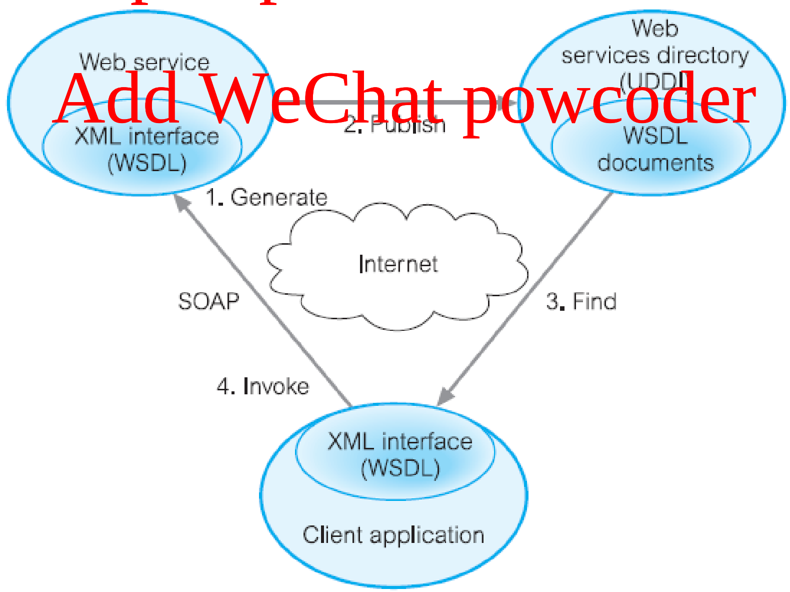
- Developers can add the Web service to a Web page (or an executable program) to offer specific functionality to users.

Web Services and Service-Oriented Architectures (2 of 3)

- Web services approach uses accepted technologies and standards, such as:
  - XML (extensible Markup Language)
  - SOAP (Simple Object Access Protocol) is a communication protocol for exchanging structured information over the Internet and uses a message format based on XML. It is both platform- and language-independent.
  - WSDL (Web Services Description Language) protocol, again based on XML, is used to describe and locate a Web service.

Web Services and Service-Oriented Architectures (3 of 3)

- UDDI (Universal Discovery, Description, and Integration) protocol is a platform independent, XML-based registry for businesses to list themselves on the Internet.



Service-Oriented Architectures (SOA)

- A business-centric software architecture for building applications that implement business processes as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published, and discovered, and are abstracted away from the implementation using a single standards-based form of interface.

Distributed DBMSs (1 of 2)

- A distributed database is a logically interrelated collection of shared data (and a description of this data), physically distributed over a computer network.
- A distributed DBMS is the software system that permits the management of the distributed database and makes the distribution transparent to users.

Distributed DBMSs (2 of 2)

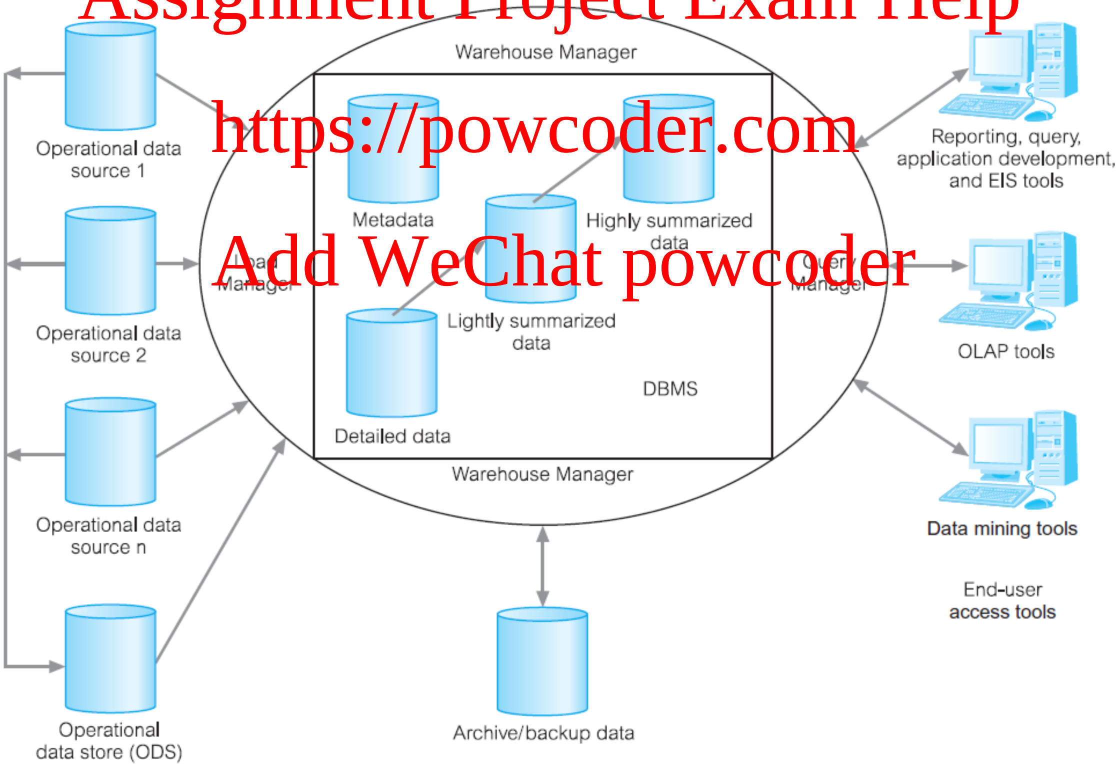
- A DDBMS consists of a single logical database split into a number **of fragments.**

- Each fragment is stored on one or more computers **(replicas)** under the control of a separate DBMS, with the computers connected by a network.

- Each site is capable of independently processing user requests that require access to local data (that is, each site has some degree of local autonomy) and is also capable of processing data stored on other computers in the network.

Data Warehousing

- A data warehouse was deemed the solution to meet the requirements of a system capable of supporting decision making, receiving data from multiple operational data sources.



Cloud Computing (2 of 2)

- The National Institute of Standards and Technology (NIST) provided a definition.
- Defined as "A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Cloud Computing – Key Characteristics (1 of 3)

- **On-demand self-service**
  – Consumers can obtain, configure and deploy cloud services without help from provider.

- **Broad network access**

  – Accessible from anywhere, from any standardized platform (e.g. desktop computers, laptops, mobile devices).

Cloud Computing – Key Characteristics (2 of 3)

- **Resource pooling**
  - Provider's computing resources are pooled to serve multiple consumers, with different physical and virtual resources dynamically assigned and reassigned according to consumer demand. Examples of resources include storage, processing, memory, and network bandwidth.

Cloud Computing – Key Characteristics (3 of 3)

- **Rapid elasticity**
  - Provider's capacity caters for customer's spikes in demand and reduces risk of outages and service interruptions. Capacity can be automated to scale rapidly based on demand.
- **Measured service**
  - Provider uses a metering capability to measure usage of service (e.g. storage, processing, bandwidth, and active user accounts).

Pearson

Cloud Computing – Service Models (1 of 3)

- **Software as a Service (SaaS):**
  - Software and data hosted on cloud. Accessed through using thin client interface (e.g. web browser). Consumer may be offered limited user specific application configuration settings.
  - Examples include Salesforce.com sales management applications, NetSuite’s integrated business management software, Google’s Gmail and Cornerstone OnDemand.

Cloud Computing – Service Models (2 of 3)

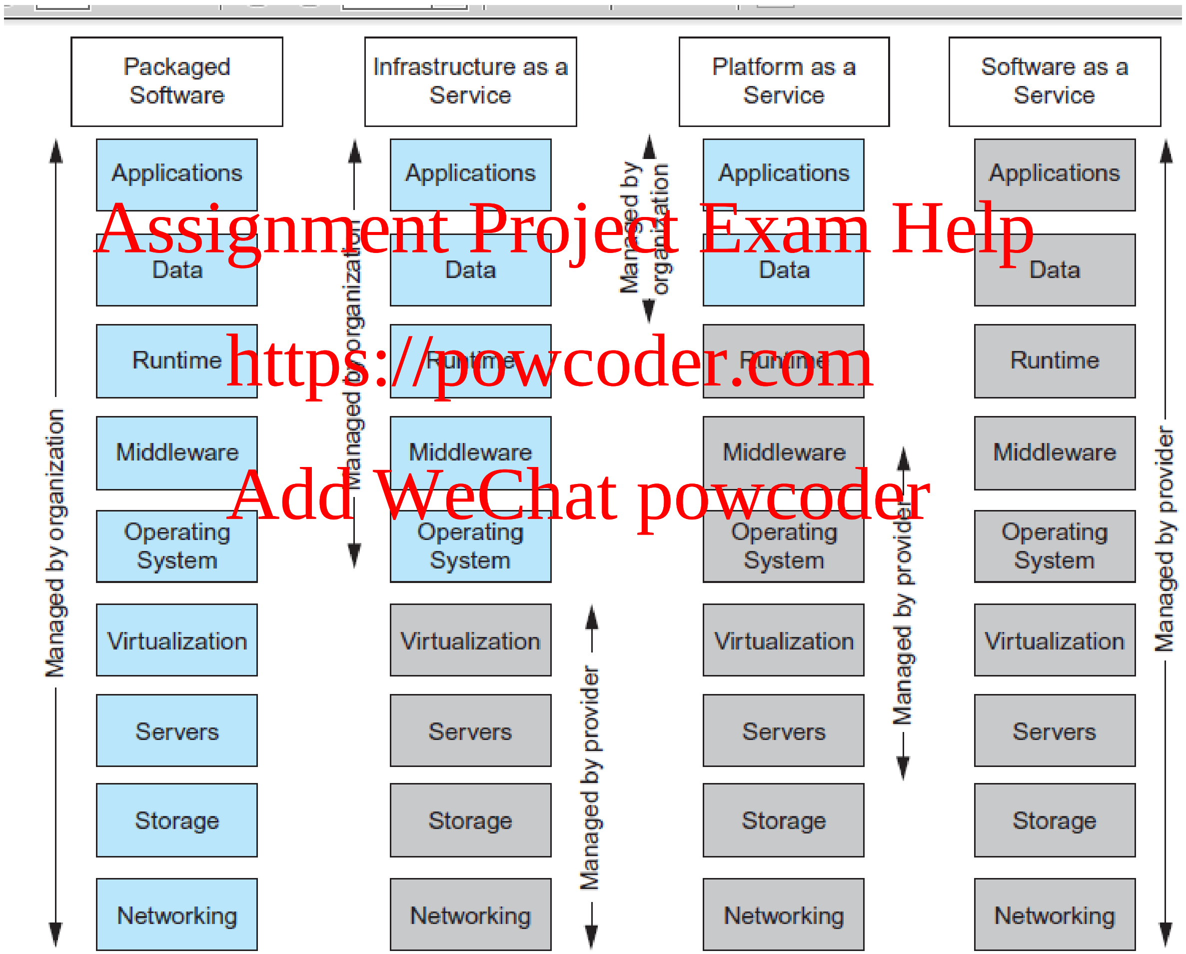
- **Platform as a Service (PaaS)**
  - Allows creation of web applications without buying/maintaining the software and underlying infrastructure. Provider manages the infrastructure including network, servers, OS and storage, while customer controls deployment of applications and possibly configuration.
  - Examples include Salesforce.com's Force.com, Google's App Engine, and Microsoft's Azure.

Pearson

Cloud Computing – Service Models (3 of 3)

- **Infrastructure as a Service (IaaS)**
  - Provider's offer servers, storage, network and operating systems – typically a platform virtualization environment – to consumers as an on-demand service, in a single bundle and billed according to usage.
  - A popular use of IaaS is in hosting websites. Examples Amazon's Elastic Compute Cloud (EC2), Rackspace and GoGrid.

Cloud Computing – Comparison of Services Models



Benefits of Cloud Computing (1 of 2)

- **Cost-Reduction:** Avoid up-front capital expenditure.
- **Scalability/Agility:** Organisations set up resources on an as-needs basis.
- **Improved Security:** Providers can devote expertise & resources to security; not affordable by customer.
- **Improved Reliability:** Providers can devote expertise & resources on reliability of systems; not affordable by customer.
- **Access to new technologies:** Through use of provider’s systems, customers may access latest technology.

Benefits of Cloud Computing (2 of 2)

- **Faster development:** Provider's platforms can provide many of the core services to accelerate development cycle.

- **Large scale prototyping/load testing:** Providers have the resources to enable this.

- **More flexible working practices:** Staff can access files using mobile devices.
- **Increased competitiveness:** Allows organizations to focus on their core competencies rather than their IT infrastructures.

Risks of Cloud Computing

- **Network Dependency:** Power outages, bandwidth issues and service interruptions.
- **System Dependency:** Customer's dependency on availability and reliability of provider's systems.
- **Cloud Provider Dependency:** Provider could became insolvent or acquired by competitor, resulting in the service suddenly terminating.
- **Lack of control:** Customers unable to deploy technical or organisational measures to safeguard the data. May result in reduced availability, integrity, confidentiality, intervenability and isolation.
- **Lack of information on processing transparency**

Cloud-Based Database Solutions (1 of 6)

- As a type of Software as a Service (SaaS), cloud-based database solutions fall into two basic categories:
  - Data as a Service (DaaS) and
  - Database as a Service (DBaaS).
- Key difference between the two options is mainly how the data is managed.

Cloud-Based Database Solutions (2 of 6)

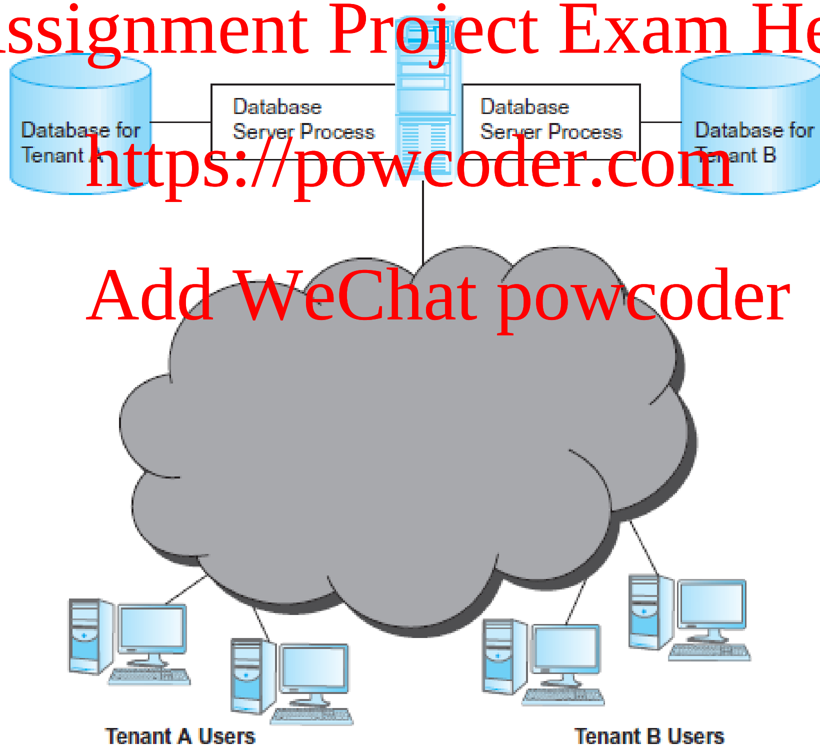
- DBaaS
  - Offers full database functionality to application developers.
  - Provides a management layer that provides continuous monitoring and configuring of the database to optimized scaling, high availability, multi-tenancy (that is, serving multiple client organizations), and effective resource allocation in the cloud, thereby sparing the developer from ongoing database administration tasks.

Cloud-Based Database Solutions (3 of 6)

- DaaS:
  - Services enables data definition in the cloud and subsequently querying.
  - Does not implement typical DBMS interfaces (e.g. SQL) but instead data is accessed via common APIs.
  - Enables organization with valuable data to offer access to others. Examples Urban Mapping (geography data service), Xignite (financial data service) and Hoovers (business data service.)

Cloud-Based Database Solutions (4 of 6)

- Multi-tenant cloud database-shared server, separate database server process architecture.

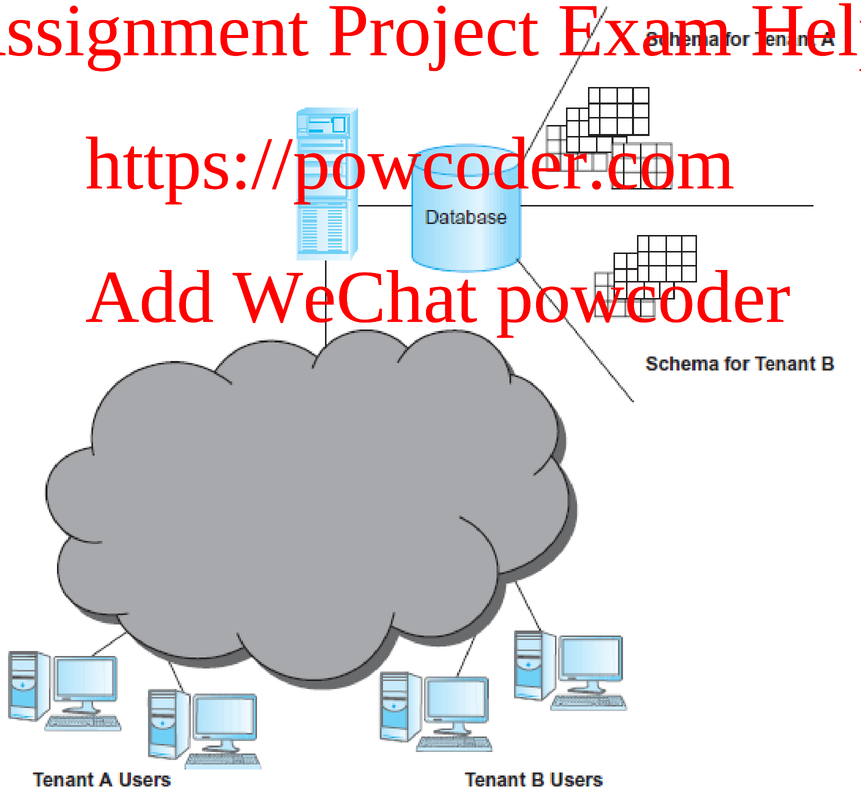


Cloud-Based Database Solutions (5 of 6)

- Multi-tenant cloud database-shared DBMS server, separate databases.

Cloud-Based Database Solutions (6 of 6)

- Multi-tenant cloud database–shared database, separate schema architecture.

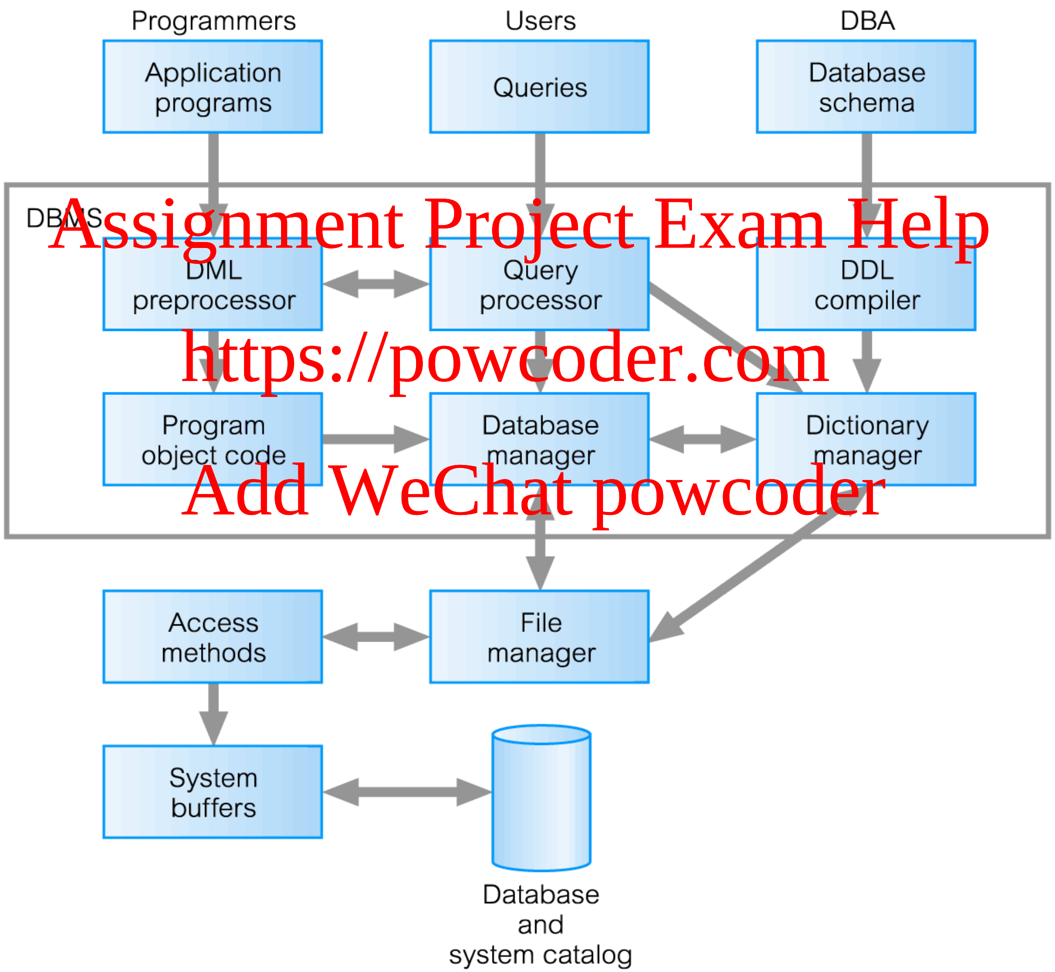


Components of a DBMS (1 of 5)

- A DBMS is partitioned into several software components (or **modules**), each of which is assigned a specific operation. As stated previously, some of the functions of the DBMS are supported by the underlying operating system.

- The DBMS interfaces with other software components, such as user queries and access methods (file management techniques for storing and retrieving data records).

Components of a DBMS (2 of 5)



Components of a DBMS (3 of 5)

- **Query processor** is a major DBMS component that transforms queries into a series of low-level instructions directed to the database manager.
- **Database manager (DM)** interfaces with user-submitted application programs and queries. The DM examines the external and conceptual schemas to determine what conceptual records are required to satisfy the request. The DM then places a call to the file manager to perform the request.

Components of a DBMS (4 of 5)

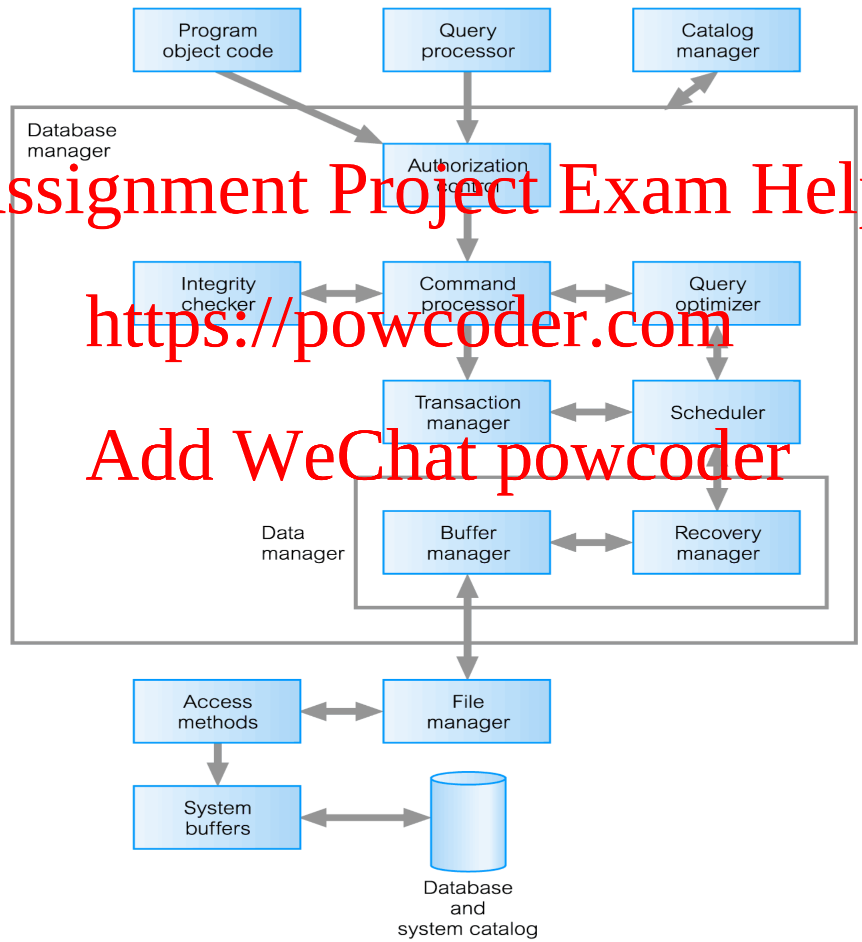
- **File manager** manipulates the underlying storage files and manages the allocation of storage space on disk. It establishes and maintains the list of structures and indexes defined in the internal schema.

- **DML preprocessor** converts DML statements embedded in an application program into standard function calls in the host language. The DML preprocessor must interact with the query processor to generate the appropriate code.

Components of a DBMS (5 of 5)

- **DDL compiler** converts DDL statements into a set of tables containing metadata. These tables are then stored in the system catalog while control information is stored in data file headers.
- **Catalog manager** manages access to and maintains the system catalog. The system catalog is accessed by most DBMS components.

Components of Database Manager (DM)



Components of the Database Manager (1 of 3)

- **Authorization control** to confirm whether the user has the necessary permission to carry out the required operation.
- **Command processor** on confirmation of user authority, control is passed to the command processor.
- **Integrity checker** ensures that requested operation satisfies all necessary integrity constraints (e.g. key constraints) for an operation that changes the database.

Components of the Database Manager (2 of 3)

- **Query optimizer** determines an optimal strategy for the query execution.
- **Transaction manager** performs the required processing of operations that it receives from transactions.
- **Scheduler** ensures that concurrent operations on the database proceed without conflicting with one another. It controls the relative order in which transaction operations are executed.

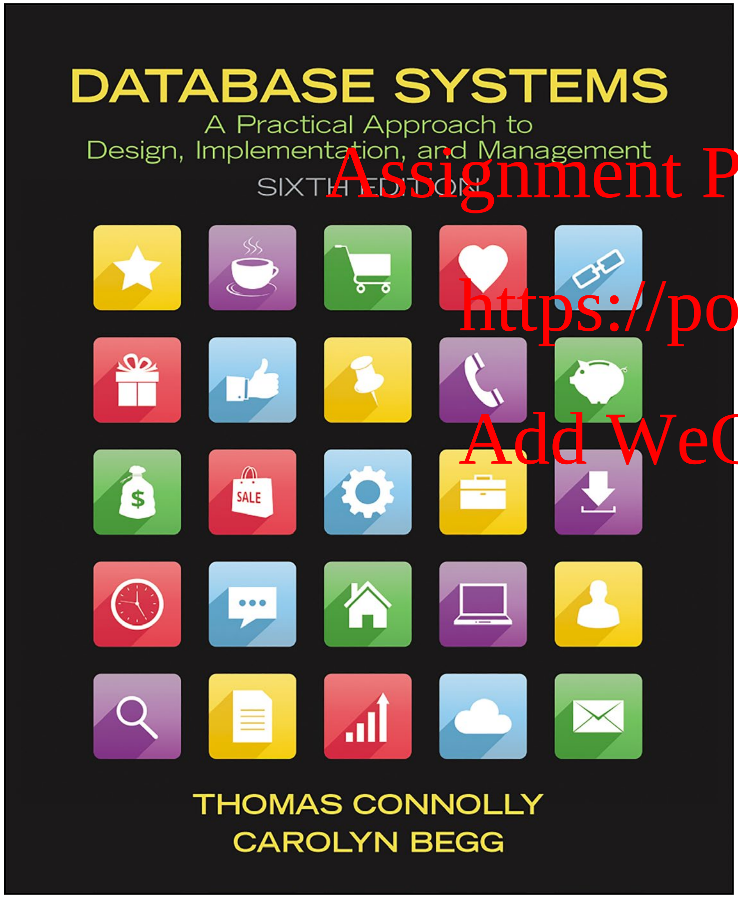
Components of the Database Manager (3 of 3)

- **Recovery manager** ensures that the database remains in a consistent state in the presence of failures. It is responsible for transaction commit and abort.
- **Buffer manager** responsible for the transfer of data between main memory and secondary storage, such as disk and tape.
- The recovery manager and the buffer manager also known as (aka) the **data manager**. The buffer manager aka the **cache manager**.

Copyright

Database Systems: A Practical Approach to Design, Implementation, and Management

Sixth Edition



Chapter 4

The Relational Model

Pearson

Learning Objectives

4.1 Terminology of relational model.

4.2 How tables are used to represent data.

4.3 Connection between mathematical relations and relations in the relational model.

4.4 Properties of database relations.

4.5 How to identify CK, PK, and FKs.

4.6 Meaning of entity integrity and referential integrity.

4.7 Purpose and advantages of views.

Relational Model Terminology (1 of 2)

- 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.

Relational Model Terminology (2 of 2)

- 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.

Instances of Branch and Staff Relations

Attributes

Branch

Relation

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

Cardinality

Degree

Primary key

Foreign key

Staff

Relation

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
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

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 1, value M or F
sex	Sex	The sex of a person	character: size 1, value M or F
DOB	DatesOfBirth	Possible value 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

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

Mathematical Definition of Relation (1 of 4)

- 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.

Mathematical Definition of Relation (2 of 4)

- 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:

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

Mathematical Definition of Relation (3 of 4)

- 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.

Mathematical Definition of Relation (4 of 4)

- Cartesian product of n sets $(D_1, D_2, \ldots, D_n)$ is:

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

usually written as:

$$\mathop{\times}_{i=1}^{n} D_i$$

- 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 (1 of 2)

- 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.

Properties of Relations (2 of 2)

- Each tuple is distinct; there are no duplicate tuples.
- Order of attributes has no significance.

- Order of tuples has no significance, theoretically.

Relational Keys (1 of 2)

- Superkey
  - An attribute, or 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.

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

Relational Keys (2 of 2)

- 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.

Integrity Constraints (1 of 3)

- 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.

Integrity Constraints (2 of 3)

- 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.

Integrity Constraints (3 of 3)

- General Constraints
  - Additional rules specified by users or database administrators that define or constrain some aspect of the enterprise.

Views (1 of 2)

- 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.

Views (2 of 2)

- A 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.
- 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 (1 of 3)

- 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.

Updating Views (2 of 3)

- 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.

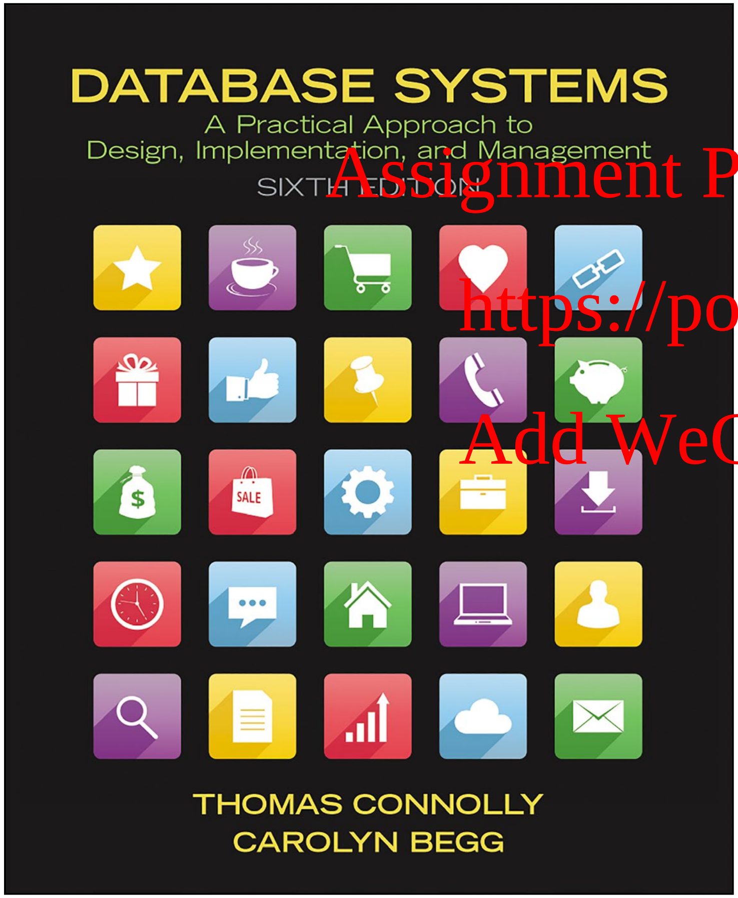
Updating Views (3 of 3)

- Classes of views are defined as:
  - theoretically not updateable;
  - theoretically updateable;
  - partially updateable.

Copyright

Database Systems: A Practical Approach to Design, Implementation, and Management

Sixth Edition



Chapter 5

Relational Algebra and Relational Calculus

Learning Objectives

5.1 Meaning of the term relational completeness.

5.2 How to form queries in relational algebra.

5.3 How to form queries in tuple relational calculus.

5.4 How to form queries in domain relational calculus.

5.5 Categories of relational DML.

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**.

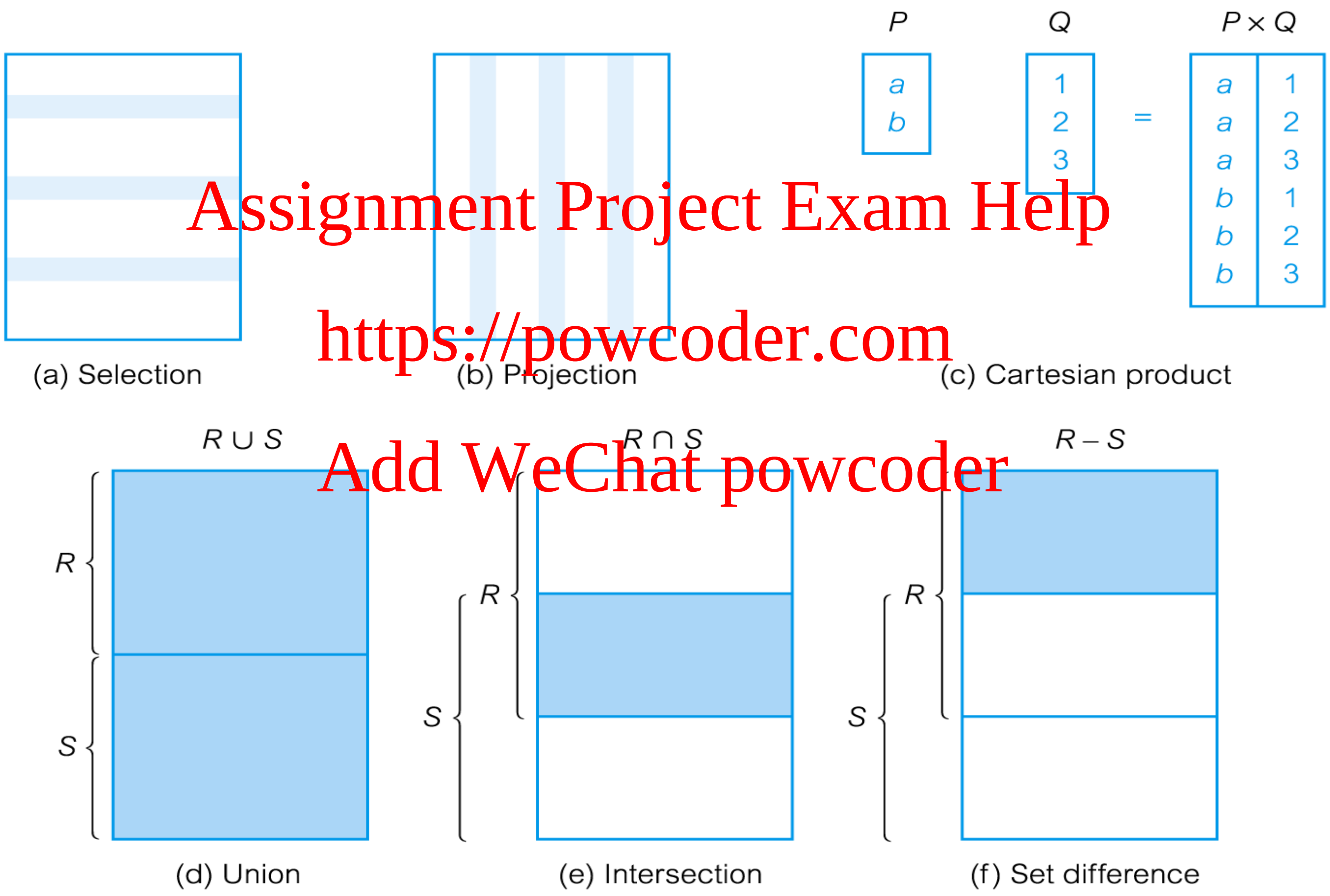
Relational Algebra (1 of 2)

- 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**.

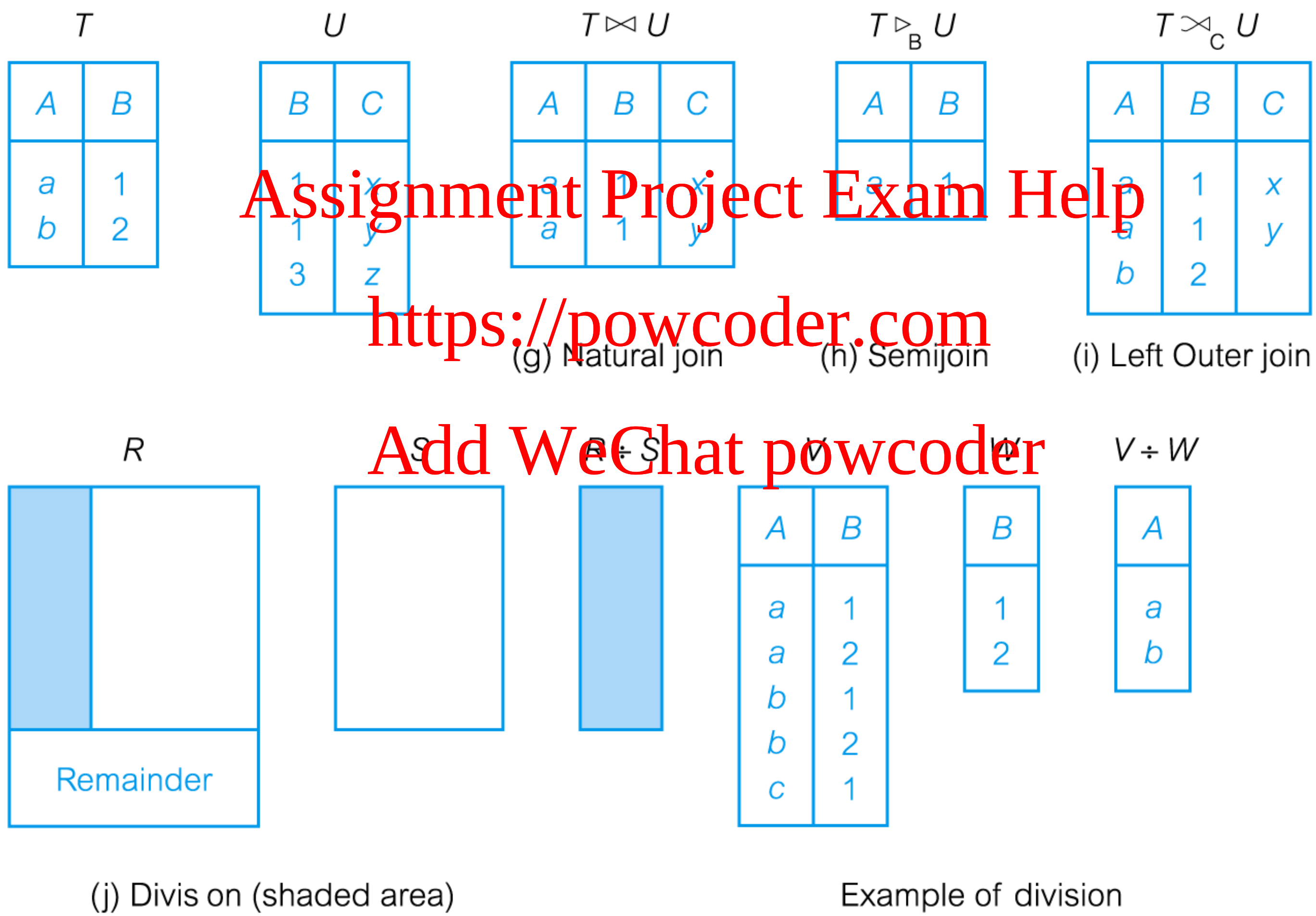
Relational Algebra (2 of 2)

- 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 (1 of 2)



Relational Algebra Operations (2 of 2)



Selection (or Restriction)

- $\sigma_{predicate}(R)$
  - 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.

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

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

- $\Pi_{\text{col1, ..., coln}}(R)$
  - 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.

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 ∪ 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

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.

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

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.

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	

Example - Cartesian Product (2 of 2)

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
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.

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

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

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

Join Operations (1 of 2)

- Join is a derivative of Cartesian product.
- 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.

Join Operations (2 of 2)

- Various forms of join operation
  - Theta join
  - Equijoin (a particular type of Theta join)
  - Natural join
  - Outer join
  - Semijoin

Theta Join (θ-Join) (1 of 2)

- $R \bowtie_F S$
  - 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 θ may be one of the comparison operators $(<, \leq, >, \geq, =, \neq)$.

Pearson

Theta Join (θ-Join) (2 of 2)

- Can rewrite Theta join using basic Selection and Cartesian product operations.

$$R \bowtie_F S = \sigma_F (R \times S)$$

- 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.

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

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

Natural Join

- $R \bowtie S$
  - 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.

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

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

Semijoin

- $R \triangleright_F S$
  - 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:

$$R \triangleright_F S \quad = \Pi_A(R \bowtie_F S)$$

Example - Semijoin

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

$$\text{Staff} \triangleright_{\text{Staff.branchNo=Branch.branchNo}}(\sigma_{\text{city='Glasgow'}}(\text{Branch}))$$

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:

$$T_1 \leftarrow \Pi_c(R)$$

$$T_2 \leftarrow \Pi_c((S \times T_1) - R)$$

$$T \leftarrow T_1 - T_2$$

Example - Division

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

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

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

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

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

Aggregate Operations

- $\mathfrak{I}_{AL}(R)$
  - Applies aggregate function list, AL, to R to define a relation over the aggregate list.
  - AL contains one or more (<aggregate_function>, <attribute>) pairs.
- Main aggregate functions are: COUNT, SUM, AVG, MIN, and MAX.

Pearson

Example – Aggregate Operations

- How many properties cost more than £350 per month to rent?

$$\rho_R(\text{myCount})\,\Im_{\text{COUNT propertyNo}}(\sigma_{\text{rent} > 350}(\text{PropertyForRent}))$$

myCount
5

(a)

Grouping Operation

- $_{GA}\Im_{AL}(R)$
  - Groups tuples of R by grouping attributes, GA, and then applies aggregate function list, AL, to define a new relation.
  - A L contains one or more (<aggregate_function>, <attribute>) pairs.
  - Resulting relation contains the grouping attributes, GA, along with results of each of the aggregate functions.

Example – Grouping Operation

- Find the number of staff working in each branch and the sum of their salaries.

$$\rho_R(\text{branchNo}, \text{myCount}, \text{mySum})$$

$${}_{\text{branchNo}}\mathfrak{I}_{\text{COUNT staffNo}, \text{SUM salary}}(\text{Staff})$$

branchNo	**myCount**	**mySum**
B003	3	54000
B005	2	39000
B007	1	9000

Relational Calculus (1 of 2)

- 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.

Relational Calculus (2 of 2)

- 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 (1 of 6)

- 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:

$$\{S \mid Staff(S) \wedge S.salary > 10000\}$$

- To find a particular attribute, such as salary, write:

$$\{S.salary \mid Staff(S) \wedge S.salary > 10000\}$$

Tuple Relational Calculus (2 of 6)

- Can use two **quantifiers** to tell how many instances the predicate applies to:
  - Existential quantifier $\exists$ ('there exists')
  - Universal quantifier $\forall$ ('for all')
- Tuple variables qualified by $\forall$ or $\exists$ are called **bound** variables, otherwise called **free** variables.

Tuple Relational Calculus (3 of 6)

- Existential quantifier used in formulae that must be true for at least one instance, such as:

$$\text{Staff(S)} \wedge (\exists \text{B})(\text{Branch(B)} \wedge (\text{B.branchNo} = \text{S.branchNo}) \wedge \text{B.city} = \text{'London'})$$

- Means 'There exists a Branch tuple with same branchNo as the branchNo of the current Staff tuple, **S**, and is located in London'.

Tuple Relational Calculus (4 of 6)

- Universal quantifier is used in statements about every instance, such as:

  $(\forall B)(B.city \neq \text{'Paris'})$

- Means ‘For all Branch tuples, the address is not in Paris’.

- Can also use $\sim(\exists B)\ (B.city = \text{‘Paris’})$ which means ‘There are no branches with an address in Paris’.

Tuple Relational Calculus (5 of 6)

- Formulae should be unambiguous and make sense.
- A (well-formed) formula is made out of **atoms**:
  - $R(S_i)$, where S_i is a tuple variable and R is a relation
  - $S_i.a_1 \ \theta S_j.a_2$
  - $S_i.a_1 \ \theta c$
- Can recursively build up formulae from atoms:
  - An atom is a formula
  - If F_1 and F_2 are formulae, so are their conjunction, $F_1 \wedge F_2$; disjunction, $F_1 \vee F_2$; and negation, $\sim F_1$
  - If F is a formula with free variable X, then $(\exists X)(F)$ and $(\forall X)(F)$ are also formulae.

Example - Tuple Relational Calculus (1 of 3)

- List the names of all managers who earn more than £25,000.

$$\{S.fName, S.lName \mid Staff(S) \wedge$$
$$S.position = 'Manager' \wedge S.salary > 25000\}$$

- List the staff who manage properties for rent in Glasgow.

$$\{S \mid Staff(S) \wedge (\exists P)(PropertyForRent(P) \wedge$$
$$(P.StaffNo = S.staffNo) \wedge P.city = 'Glasgow')\}$$

Example - Tuple Relational Calculus (2 of 3)

- List the names of staff who currently do not manage any properties.

$$\{S.fName, S.lName \mid Staff(S) \wedge (\sim(\exists P)$$

$$(PropertyForRent(P) \wedge (S.staffNo = P.staffNo)))\}$$

Or

$$\{S.fName, S.lName \mid Staff(S) \wedge ((\forall P)$$

$$(\sim PropertyForRent(P) \vee$$

$$\sim (S.staffNo = P.staffNo)))\}$$

Example - Tuple Relational Calculus (3 of 3)

- List the names of clients who have viewed a property for rent in Glasgow.

$$\{C.fName, C.lName \mid Client(C) \wedge ((\exists V)(\exists P)$$
$$(Viewing(V) \wedge PropertyForRent(P) \wedge$$
$$(C.clientNo = V.clientNo) \wedge$$
$$(V.propertyNo = P.propertyNo) \wedge$$
$$P.city = \text{‘Glasgow’}))\}$$

Tuple Relational Calculus (6 of 6)

- Expressions can generate an infinite set. For example:

  {S |~ Staff(S)}

- To avoid this, add restriction that all values in result must be values in the domain of the expression.

Domain Relational Calculus (1 of 2)

- Uses variables that take values from **domains** instead of tuples of relations.
- If F $(d_1, d_2, \ldots, d_n)$ stands for a formula composed of composed of atoms and $d_1, d_2, \ldots, d_n$ represent domain, then:

$$\{d_1, d_2, \ldots, d_n \mid F(d_1, d_2, \ldots, d_n)\}$$

is a general domain relational calculus expression.

Example - Domain Relational Calculus (1 of 4)

- Find the names of all managers who earn more than £25,000.

$$\{fN, lN \mid (\exists sN, posn, sex, DOB, sal, bN)$$
$$(Staff(sN, fN, lN, posn, sex, DOB, sal, bN) \wedge$$
$$posn = \text{'Manager'} \wedge sal > 25000)\}$$

Example - Domain Relational Calculus (2 of 4)

- List the staff who manage properties for rent in Glasgow.

$$\{sN, fN, lN, posn, sex, DOB, sal, bN \mid (\exists sN1, cty)(Staff(sN, fN, lN, posn, sex, DOB, sal, bN) \wedge PropertyForRent(pN, st, cty, pc, typ, rms, rnt, oN, sN1, bN1) \wedge (sN = sN1) \wedge cty = \text{'Glasgow'})\}$$

Example - Domain Relational Calculus (3 of 4)

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

$$\{fN, lN \mid (\exists sN)$$
$$(Staff(sN, fN, lN, posn, sex, DOB, sal, bN) \wedge$$
$$(\sim (\exists sN1)(PropertyForRent(pN, st, cty, pc, typ,$$
$$rms, rnt, oN, sN1, bN1) \wedge (sN = sN1))))\}$$

Example - Domain Relational Calculus (4 of 4)

- List the names of clients who have viewed a property for rent in Glasgow.

$$\{fN, lN \mid (\exists cN, cN1, pN, pN1, cty)$$

$$(Client(cN, fN, lN, tel, pT, mR) \wedge$$

$$Viewing(cN1, pN1, dt, cmt) \wedge$$

$$PropertyForRent(pN, st, cty, pc, typ, rms, rnt, oN, sN, bN) \wedge$$

$$(cN = cN1) \wedge (pN = pN1) \wedge cty = \text{'Glasgow'})\}$$

Domain Relational Calculus (2 of 2)

- When restricted to safe expressions, domain relational calculus is equivalent to tuple relational calculus restricted to safe expressions, which is equivalent to relational algebra.
- Means every relational algebra expression has an equivalent relational calculus expression, and vice versa.

Other Languages (1 of 2)

- Transform-oriented languages are non-procedural languages that use relations to transform input data into required outputs (e.g. SQL).
- Graphical languages provide user with picture of the structure of the relation. User fills in example of what is wanted and system returns required data in that format (e.g. QBE).

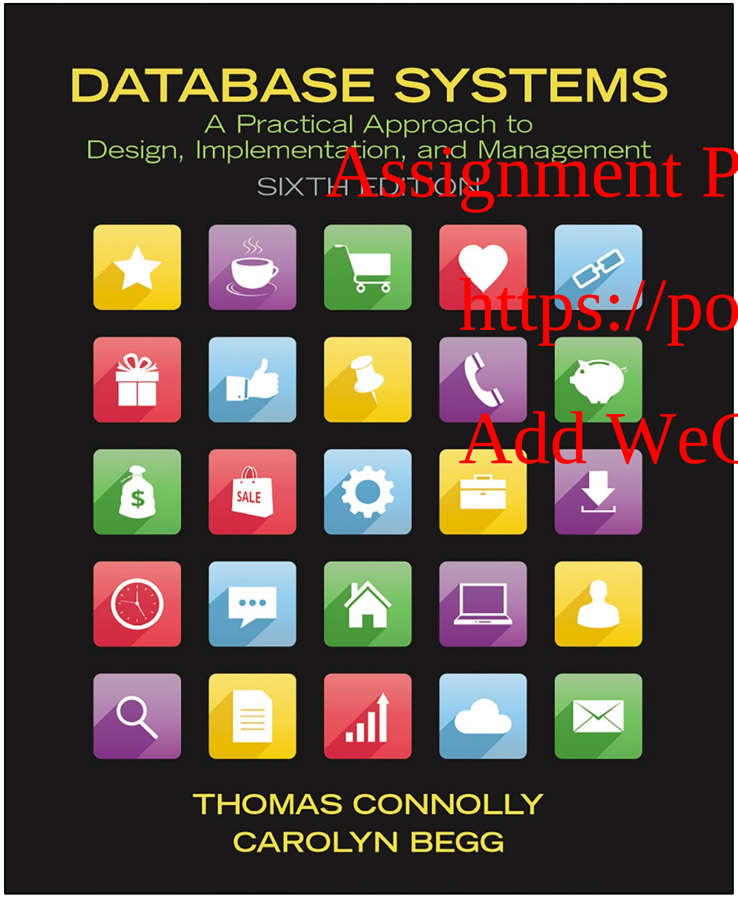
Other Languages (2 of 2)

- 4GLs can create complete customized application using limited set of commands in a user-friendly, often menu-driven environment.
- Some systems accept a form of **natural language**, sometimes called a 5GL, although this development is still at an early stage.

Copyright

Database Systems: A Practical Approach to Design, Implementation, and Management

Sixth Edition



Chapter 6

SQL – Data Manipulation

Learning Objectives (1 of 2)

6.1 Purpose and importance of SQL.

6.2 How to retrieve data from database using SELECT and:

– Use compound WHERE conditions.

– Sort query results using ORDER BY.

– Use aggregate functions.

– Group data using GROUP BY and HAVING.

– Use subqueries.

Learning Objectives (2 of 2)

- Join tables together.
- Perform set operations (UNION, INTERSECT, EXCEPT).

6.3 How to update database using INSERT, UPDATE, and DELETE.

Objectives of SQL (1 of 5)

- 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.

Objectives of SQL (2 of 5)

- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.
- Until SQL:1999, 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.

Objectives of SQL (3 of 5)

- 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.

Objectives of SQL

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;
```

Objectives of SQL (5 of 5)

- 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.

History of SQL (1 of 3)

- 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.

History of SQL (2 of 3)

- 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.

History of SQL (3 of 3)

- 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, SQL:1999 was released with support for object-oriented data management.
- In late 2003, SQL:2003 was released.
- In summer 2008, SQL:2008 was released.
- In late 2011, SQL:2011 was released.

Importance of SQL (1 of 2)

- 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.

Importance of SQL (2 of 2)

- 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 (1 of 3)

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

Writing SQL Commands (2 of 3)

- 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.

Writing SQL Commands (3 of 3)

- 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).

SELECT Statement (1 of 3)

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

SELECT Statement (2 of 3)

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.

SELECT Statement (3 of 3)

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

Example 6.1 All Columns, All Rows (1 of 2)

List full details of all staff.

```
SELECT staffNo, fName, lName, address,
position, sex, DOB, salary, branchNo
FROM Staff;
```


- Can use * as an abbreviation for ‘all columns’:

```
SELECT *
FROM Staff;
```

Example 6.1 All Columns, All Rows (2 of 2)

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

Example 6.2 Specific Columns, All Rows (1 of 2)

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;
```

Example 6.2 Specific Columns, All Rows (2 of 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

Example 6.3 Use of DISTINCT (1 of 2)

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


```
SELECT propertyNo
FROM Viewing;
```


propertyNo
PA14
PG4
PG4
PA14
PG36

Example 6.3 Use of DISTINCT (2 of 2)

- Use DISTINCT to eliminate duplicates:

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


propertyNo
PA14
PG4
PG36

Example 6.4 Calculated Fields (1 of 2)

Produce list of monthly salaries for all staff, showing staff number, first/last name, and salary.


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


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

Example 6.4 Calculated Fields (2 of 2)

- To name column, use AS clause:


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

Example 6.5 Comparison Search Condition

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

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


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

Example 6.6 Compound Comparison Search Condition

List addresses of all branch offices in London or Glasgow.

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


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

Example 6.7 Range Search Condition (1 of 3)

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.

Example 6.7 Range Search Condition (2 of 3)

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

Example 6.7 Range Search Condition (3 of 3)

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


- Useful, though, for a range of values.

Example 6.8 Set Membership (1 of 2)

List all managers and supervisors.

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


staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Example 6.8 Set Membership (2 of 2)

- 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.

Example 6.9 Pattern Matching (1 of 2)

Find all owners with the string ‘Glasgow’ in their address.

```
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE ‘%Glasgow%’;
```


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

Example 6.9 Pattern Matching (2 of 2)

- 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’.

Example 6.10 NULL Search Condition (1 of 2)

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

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

Example 6.10 NULL Search Condition (2 of 2)

clientNo	viewDate
CR 56	26-May-13

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

Example 6.11 Single Column Ordering (1 of 2)

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


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

Example 6.11 Single Column Ordering (2 of 2)

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

Example 6.12 Multiple Column Ordering (1 of 4)

Produce abbreviated list of properties in order of property type.


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

Example 6.12 Multiple Column Ordering (2 of 4)

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

Example 6.12 Multiple Column Ordering (3 of 4)

- 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;
```

Example 6.12 Multiple Column Ordering (4 of 4)

propertyNo	**type**	**rooms**	**rent**
PL16	Flat	4	450
PG94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

SELECT Statement – Aggregates (1 of 4)

- 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.

SELECT Statement – Aggregates (2 of 4)

- 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.

SELECT Statement – Aggregates (3 of 4)

- 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.

SELECT Statement – Aggregates (4 of 4)

- 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;
```

Example 6.13 Use of COUNT(*)

- How many properties cost more than £350 per month to rent?


```
SELECT COUNT(*) AS myCount
FROM PropertyForRent
WHERE rent > 350;
```


myCount
5

Example 6.14 Use of COUNT(DISTINCT)

- How many different properties viewed in May '13?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN '1-May-13'
        AND '31-May-13';
```


myCount
2

Example 6.15 Use of COUNT and SUM

- Find number of Managers and sum of their salaries.

```
SELECT COUNT(staffNo) AS myCount,
SUM(salary) AS mySum
FROM Staff
WHERE position = ‘Manager’;
```


myCount	mySum
2	54000.00

Example 6.16 Use of MIN, MAX, AVG

- Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS myMin,
       MAX(salary) AS myMax,
       AVG(salary) AS myAvg
FROM Staff;
```


myMin	myMax	myAvg
9000.00	30000.00	17000.00

SELECT Statement – Grouping (1 of 2)

- 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 clause may only contain:
  - column names
  - aggregate functions
  - constants
  - expression involving combinations of the above.

SELECT Statement – Grouping (2 of 2)

- 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.

Example 6.17 Use of GROUP BY (1 of 2)

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

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

Example 6.17 Use of GROUP BY (2 of 2)

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

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.

Example 6.18 Use of HAVING (1 of 2)

- 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 myCount,
        SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

Example 6.18 Use of HAVING (2 of 2)

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

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.

Example 6.19 Subquery with Equality (1 of 3)

- List staff who work in branch at ‘163 Main St’.

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

Example 6.19 Subquery with Equality (2 of 3)

- 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’;
```

Example 6.19 Subquery with Equality (3 of 3)

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

Example 6.20 Subquery with Aggregate (1 of 3)

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);
```

Example 6.20 Subquery with Aggregate

- 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:


```
SELECT staffNo, fName, lName, position,
salary – 17000 As salDiff
FROM Staff
WHERE salary > 17000;
```

Example 6.20 Subquery with Aggregate (3 of 3)

staffNo	fName	lName	position	salDiff
SL21	John	White	Manager	13000.00
SG14	David	Ford	Supervisor	1000.00
SG5	Susan	Brand	Manager	7000.00

Subquery Rules (1 of 2)

- 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**.

Subquery Rules (2 of 2)

- 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.

Example 6.21 Nested Subquery: Use of IN (1 of 2)

- 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'));
```

Example 6.21 Nested Subquery: Use of IN (2 of 2)

propertyNo	street	city	postcode	type	rooms	rent
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375
PG21	18 Dale Rd	Glasgow	G12	House	5	600

ANY and ALL

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- With ALL, condition will only be true if it is satisfied by **all** values produced by subquery.
- With ANY, condition will be true if it is satisfied by **any** values produced by subquery.
- If subquery is empty, ALL returns true, ANY returns false.
- SOME may be used in place of ANY.

Example 6.22 Use of ANY/SOME (1 of 2)

- Find staff whose salary is larger than salary of at least one member of staff at branch B003.


```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > SOME
        (SELECT salary
        FROM Staff
        WHERE branchNo = 'B003');
```

Example 6.22 Use of ANY/SOME (2 of 2)

- Inner query produces set {12000, 18000, 24000} and outer query selects those staff whose salaries are greater than any of the values in this set.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Example 6.23 Use of ALL (1 of 2)

- Find staff whose salary is larger than salary of every member of staff at branch B003.


```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > ALL
      (SELECT salary
      FROM Staff
      WHERE branchNo = ‘B003’);
```

Example 6.23 Use of ALL (2 of 2)

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

Multi-Table Queries (1 of 2)

- 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.
- Use comma as separator and typically include WHERE clause to specify join column(s).

Multi-Table Queries (2 of 2)

- 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.

Example 6.24 Simple Join (1 of 2)

- 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;
```

Example 6.24 Simple Join (2 of 2)

- 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.

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

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

Example 6.25 Sorting a Join (1 of 2)

- 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;
```

Example 6.25 Sorting a Join (2 of 2)

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

Example 6.26 Three Table Join (1 of 2)

- 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;
```

Example 6.26 Three Table Join (2 of 2)

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

Example 6.27 Multiple Grouping Columns (1 of 2)

- Find number of properties handled by each staff member.

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS
    myCount
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

Example 6.27 Multiple Grouping Columns (2 of 2)

branchNo	staffNo	myCount
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Computing a Join (1 of 2)

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.

Computing a Join (2 of 2)

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 (1 of 3)

- 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:

Branch 1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent 1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Outer Joins (2 of 3)

- The (inner) join of these two tables:

```
SELECT b.*, p.*
FROM Branch1 b, PropertyForRent1 p
WHERE b.bCity = p.pCity;
```


branchNo	**bCity**	**PropertyNo**	**pCity**
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Outer Joins (3 of 3)

- 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 join.

Example 6.28 Left Outer Join (1 of 2)

- 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;
```

Example 6.28 Left Outer Join (2 of 2)

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

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

Example 6.29 Right Outer Join (1 of 2)

- 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;
```

Example 6.29 Right Outer Join (2 of 2)

- 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.

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

Example 6.30 Full Outer Join (1 of 2)

- 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;
```

Example 6.30 Full Outer Join (2 of 2)

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

branchNo	**bCity**	**propertyNo**	**pCity**
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

EXISTS and NOT EXISTS (1 of 2)

- EXISTS and NOT EXISTS are for use only with subqueries.
- Produce a simple true/false result.
- True if and only if there exists at least one row in result table returned by subquery.
- False if subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.

EXISTS and NOT EXISTS (2 of 2)

- As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.

- Common for subqueries following (NOT) EXISTS to be of form:

(SELECT *…)

Example 6.31 Query using EXISTS (1 of 4)

- Find all staff who work in a London branch.

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

Example 6.31 Query using EXISTS (2 of 4)

staffNo	fName	lName	position
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

Example 6.31 Query using EXISTS (3 of 4)

- Note, search condition s.branchNo = b.branchNo is necessary to consider correct branch record for each member of staff.

- If omitted, would get all staff records listed out because subquery:


```
SELECT * FROM Branch WHERE city='London'
```

- would always be true and query would be:

```
SELECT staffNo, fName, lName, position FROM Staff
WHERE true;
```

Example 6.31 Query using EXISTS (4 of 4)

- Could also write this query using join construct:

```
SELECT staffNo, fName, lName, position
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo AND
    city = 'London';
```

Union, Intersect, and Difference (Except) (1 of 3)

- Can use normal set operations of Union, Intersection, and Difference to combine results of two or more queries into a single result table.

- Union of two tables, A and B, is table containing all rows in either A or B or both.

- Intersection is table containing all rows common to both A and B.
- Difference is table containing all rows in A but not in B.
- Two tables must be **union compatible**.

Union, Intersect, and Difference (Except) (2 of 3)

- Format of set operator clause in each case is:

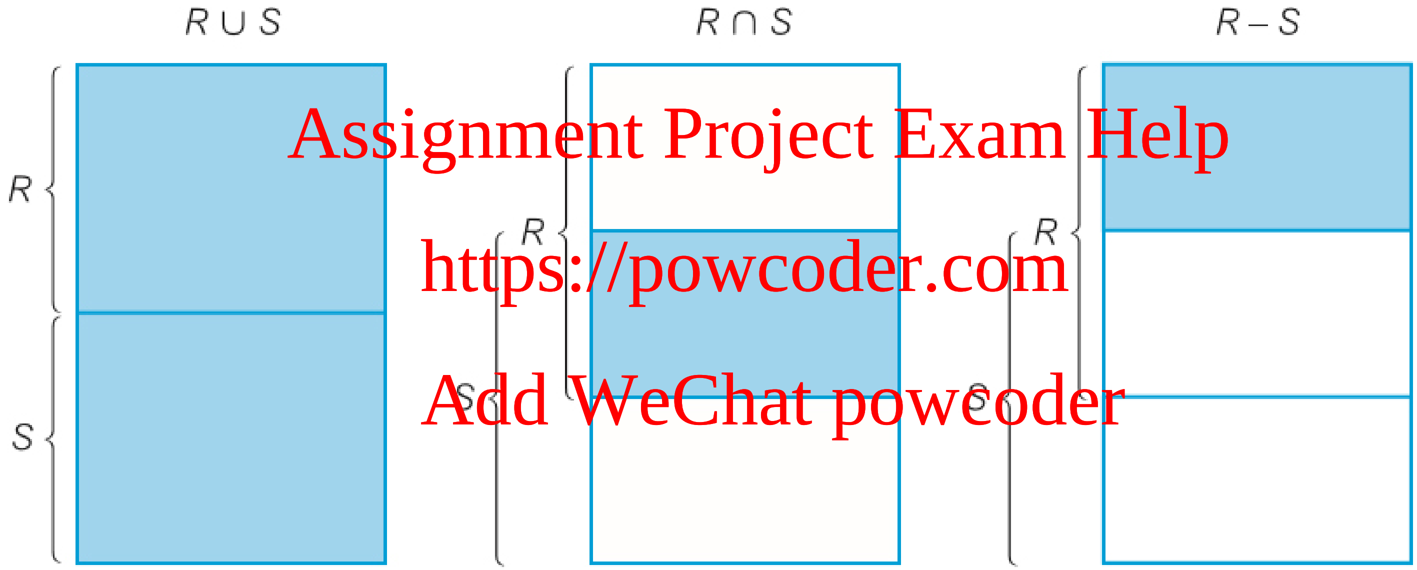
  op [ALL] [CORRESPONDING [BY {column1 [, ...]}]]

- If CORRESPONDING BY specified, set operation performed on the named column(s).

- If CORRESPONDING specified but not BY clause, operation performed on common columns.

- If ALL specified, result can include duplicate rows

Union, Intersect, and Difference (Except) (3 of 3)



(a) Union (b) Intersection (c) Difference

Example 6.32 Use of UNION (1 of 3)

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


```
(SELECT city
FROM Branch
WHERE city IS NOT NULL) UNION
(SELECT city
FROM PropertyForRent
WHERE city IS NOT NULL);
```

Example 6.32 Use of UNION (2 of 3)

- Or

```
(SELECT *
FROM Branch
WHERE city IS NOT NULL)
UNION CORRESPONDING BY city
(SELECT *
FROM PropertyForRent
WHERE city IS NOT NULL);
```

Example 6.32 Use of UNION (3 of 3)

- Produces result tables from both queries and merges both tables together.

city
London
Glasgow
Aberdeen
Bristol

Example 6.33 Use of INTERSECT (1 of 3)

- List all cities where there is both a branch office and a property.


```
(SELECT city FROM Branch)
INTERSECT
(SELECT city FROM PropertyForRent);
```

Example 6.33 Use of INTERSECT (2 of 3)

- Or

```
(SELECT * FROM Branch)
INTERSECT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```


city
Aberdeen
Glasgow
London

Example 6.33 Use of INTERSECT (3 of 3)

- Could rewrite this query without INTERSECT operator:

```
SELECT b.city
FROM Branch b PropertyForRent p
WHERE b.city = p.city;
```


- Or:

```
SELECT DISTINCT city FROM Branch b
WHERE EXISTS
    (SELECT * FROM PropertyForRent p
    WHERE p.city = b.city);
```

Example 6.34 Use of EXCEPT (1 of 2)

- List of all cities where there is a branch office but no properties.

```
(SELECT city FROM Branch)
EXCEPT
(SELECT city FROM PropertyForRent);
```


- Or

```
(SELECT * FROM Branch)
EXCEPT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

city
Bristol

Example 6.34 Use of EXCEPT (2 of 2)

- Could rewrite this query without EXCEPT:

```
SELECT DISTINCT city FROM Branch
WHERE city NOT IN
    (SELECT city FROM PropertyForRent);
```


- Or

```
SELECT DISTINCT city FROM Branch b
WHERE NOT EXISTS
    (SELECT * FROM PropertyForRent p
    WHERE p.city = b.city);
```

INSERT (1 of 2)

INSERT INTO TableName [(columnList)]
VALUES (dataValueList)

- **columnList** is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.

- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.

INSERT (2 of 2)

- **dataValueList** must match **columnList** as follows:
  - number of items in each list must be same;
  - must be direct correspondence in position of items in two lists;
  - data type of each item in **dataValueList** must be compatible with data type of corresponding column.

Example 6.35 INSERT … VALUES

- Insert a new row into Staff table supplying data for all columns.


```
INSERT INTO Staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M',
        Date'1957-05-25', 8300, 'B003');
```

Example 6.36 INSERT using Defaults

- Insert a new row into Staff table supplying data for all mandatory columns.


```
INSERT INTO Staff (staffNo, fName, lName,
                   position, salary, branchNo)
VALUES ('SG44', 'Anne', 'Jones',
        'Assistant', 8100, 'B003');
```

- Or

```
INSERT INTO Staff
VALUES ('SG44', 'Anne', 'Jones', 'Assistant', NULL,
        NULL, 8100, 'B003');
```

INSERT … SELECT

- Second form of INSERT allows multiple rows to be copied from one or more tables to another:


```
INSERT INTO TableName [ (columnList) ]
    SELECT ...
```

Example 6.37 INSERT … SELECT (1 of 3)

Assume there is a table StaffPropCount that contains names of staff and number of properties they manage:

StaffPropCount(**staffNo**, fName, lName, propCnt)

Populate StaffPropCount using Staff and PropertyForRent tables.

Example 6.37 INSERT … SELECT (2 of 3)

```
INSERT INTO StaffPropCount
    (SELECT s.staffNo, fName, lName, COUNT(*)
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo
    GROUP BY s.staffNo, fName, lName)
    UNION
    (SELECT staffNo, fName, lName, 0
    FROM Staff
    WHERE staffNo NOT IN
        (SELECT DISTINCT staffNo
        FROM PropertyForRent));
```

Example 6.37 INSERT … SELECT (3 of 3)

staffNo	fName	lName	propCount
SG14	David	Ford	1
SL21	John	White	0
SG37	Ann	Beech	2
SA9	Mary	Howe	1
SG5	Susan	Brand	0
SL41	Julie	Lee	1

- If second part of UNION is omitted, excludes those staff who currently do not manage any properties.

UPDATE (1 of 2)

```
UPDATE TableName
SET columnName1 = dataValue1
        [, columnName2 = dataValue2...]
[WHERE searchCondition]
```


- **TableName** can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.

UPDATE (2 of 2)

- WHERE clause is optional:
  - if omitted, named columns are updated for all rows in table;
  - If specified, only those rows that satisfy **searchCondition** are updated.
- New **dataValue(s)** must be compatible with data type for corresponding column.

Example 6.38/39 UPDATE All Rows

Give all staff a 3% pay increase.

```
UPDATE Staff
SET salary = salary*1.03;
```


Give all Managers a 5% pay increase.


```
UPDATE Staff
SET salary = salary*1.05
WHERE position = ‘Manager’;
```

Example 6.40 UPDATE Multiple Columns

Promote David Ford (staffNo=‘SG14’) to Manager and change his salary to £18,000.


```
UPDATE Staff
SET position = ‘Manager’, salary = 18000
WHERE staffNo = ‘SG14’;
```

DELETE

```
DELETE FROM TableName
[WHERE searchCondition]
```


- **TableName** can be name of a base table or an updatable view.

- **searchCondition** is optional; if omitted, all rows are deleted from table. This does not delete table. If **search_condition** is specified, only those rows that satisfy condition are deleted.

Example 6.41/42 DELETE Specific Rows

Delete all viewings that relate to property PG4.

```
DELETE FROM Viewing
WHERE propertyNo = 'PG4';
```


Delete all records from the Viewing table.


```
DELETE FROM Viewing;
```

Copyright