Words of Concern

Point to be As Allan Bloom has said "Education is the movement from darkness to light". Through this handbook, I have tried to illuminate what might otherwise appear as black boxes to some. In doing so, I have used references from several other authors to synthesize or simplify or elaborate information. This is not possible without omitting details that I deem trivial while dilating the data that I consider relevant to topic. Every effort has been made to avoid errors. Inspite of this, some errors might have crept in. Any errors or discrepancies noted maybe brought to my notice which I shall rectify in my next revision.

Thanks,
Raghu Gurumurthy
Interface College of Computer Applications (ICCA)
Davangere

Bibliography

Ramez Elamassri, Shankant B. Navathe, Fundamentals of Database Systems, Pearson, 7th Edition, 2015
Bipin Desai, An Introduction to database systems, Galgotia Publications,2010.
C J Date: Introduction to Database System
Abraham Silberschatz, Henry Korth, S.Sudarshan, Database Systems Concepts, Sixth Edition, McGraw Hill, 2010.
Raghu Rama krishnan and Johannes Gehrke, Database management systems, Third Edition, 2002

Syllabus

Course Title: Database Management System
Course code: 21BCA3C7L
Total Contact Hours: 42
Course Credits: 03
Formative Assessment Marks: 40
Duration of SEE/Exam: 03 Hours
Summative Assessment Marks: 60

Course Outcomes (CO's)

At the end of the course, students will be able to:

- Explain the various database concepts and the need for database systems.
- Identify and define database objects, enforce integrity constraints on a database using DBMS.
- Demonstrate a Data model and Schemas in RDBMS.
- Identify entities and relationships and draw ER diagram for a given real-world problem.
- Convert an ER diagram to a database schema and deduce it to the desired normal form.
- Formulate queries in Relational Algebra, Structured Query Language (SQL) for database manipulation.
- Explain the transaction processing and concurrency control techniques.

DSC7: Database Management System (DBMS)

Unit – I 14 - Hours

Database Architecture: Introduction to Database system applications. Characteristics and Purpose of database approach. People associated with Database system. Data models. Database schema. Database architecture. Data independence. Database languages, interfaces, and classification of DBMS.

E-R Model: Entity-Relationship modeling: E – R Model Concepts: Entity, Entity types, Entity sets, Attributes, Types of attributes, key attribute, and domain of an attribute. Relationships between the entities. Relationship types, roles and structural constraints, degree and cardinality ratio of a relationship. Weak entity types, E -R diagram.

Unit – II 14 - Hours

Relational Data Model: Relational model concepts. Characteristics of Relations. Relational model constraints: Domain constrains, key constraints, Primary & foreign key constraints, integrity constraints and null values.

Relational Algebra: Basic Relational Algebra operations. Set theoretical operations on relations. JOIN operations Aggregate Functions and Grouping. Nested Sub Queries-Views. Introduction to PL/SQL & programming of above operations in PL/SQL

Data Normalization: Anomalies in relational database design. Decomposition. Functional dependencies. Normalization. First normal form, Second normal form, Third normal form. Boyce-Codd normal form.

Unit – III 14 - Hours

Query Processing Transaction Management: Introduction Transaction Processing. Single user & multiuser systems. Transactions: read & write operations.

Need of concurrency control: The lost update problem, Dirty read problem. Types of failures. Transaction states. Desirable properties (ACID properties) of Transactions. Concurrency Control Techniques: Locks and Time stamp Ordering. Deadlock & Starvation.

Unit I – Database Architecture

Introduction to Database System Applications

- A database is a collection of related data.
- By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. Nowadays, this data is typically stored in mobile phones, which have their own simple database software.
- This data can also be recorded in an indexed address book or stored on a hard drive, using a personal computer and software such as Microsoft Access or Excel.
- This collection of related data with an implicit meaning is a database.

The preceding definition of database is quite general; for example, we may consider the collection of words that make up this page of text to be related data and hence to constitute a database. However, the common use of the term database is usually more restricted.
A database has the following implicit properties

- A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

Definition

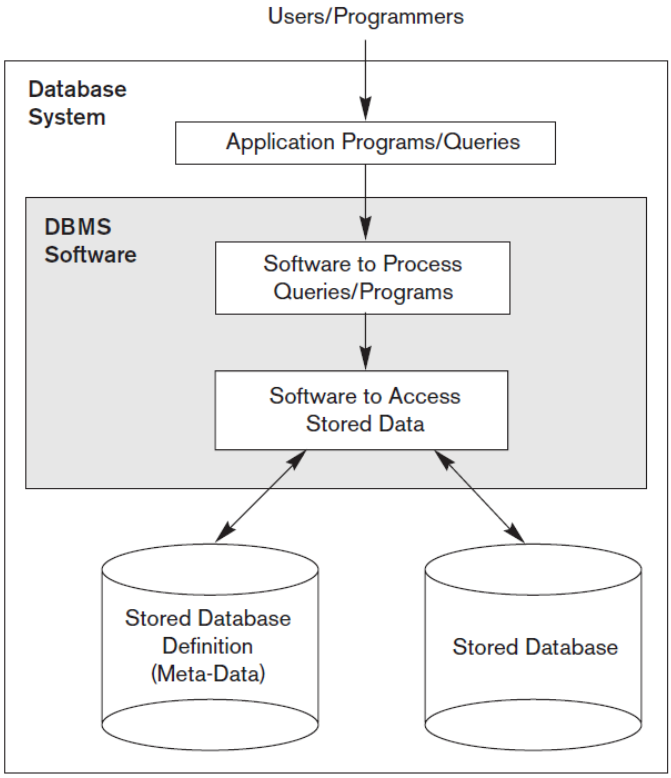
A database management system (DBMS) is a computerized system that enables users to create and maintain a database. The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing databases among various users and applications.

Closer look at Database Creation, Accessing, Manipulation, Protection and Maintenance

- Defining a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called meta-data.
- Constructing the database is the process of storing the data on some storage medium that is controlled by the DBMS. Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
- Sharing a database allows multiple users and programs to access the database simultaneously.
- An application program accesses the database by sending queries or requests for data to the DBMS.
- A query typically causes some data to be retrieved; a transaction may cause some data to be read and some data to be written into the database.

- Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time.
- Protection includes system protection against hardware or software malfunction (or crashes) and security protection against unauthorized or malicious access.
- A typical large database may have a life cycle of many years, so the DBMS must be able to maintain the database system by allowing the system to evolve as requirements change over time.

Below figure illustrates some of the concepts we have discussed so far.



Characteristics and Purpose of database approach

- A number of characteristics distinguish the database approach from the much older approach of writing customized programs to access data stored in files. In traditional file processing, each user defines and implements the files needed for a specific software application as part of programming the application.
- Duplication or Redundancy of data is the main issue with respect to the file system.
- This redundancy in defining and storing data results in wasted storage space and in redundant efforts to maintain common up-to-date data.
- In the database approach, a single repository maintains data that is defined once and then accessed by various users repeatedly through queries, transactions, and application programs.

The main characteristics of the database approach versus the file-processing approach are the following

- Self-describing nature of a database system
- Insulation between programs and data, and data abstraction
- Support of multiple views of the data
- Sharing of data and multiuser transaction processing

Self-describing nature of a database system

- A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.
- This definition is stored in the DBMS **catalog**, which contains information such as the structure of each file, the type and storage format of each data item, and various constraints on the data.
- The information stored in the catalog is called **meta-data**, and it describes the structure of the primary database.
- The catalog is used by the DBMS software and also by database users who need information about the database structure.

Below tables shows the example of a database catalog for the database.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
….	….	…..
….	….	…..
….	….	…..
Prerequisite_number	XXXXNNNN	PREREQUISITE

Insulation between programs and data, and data abstraction

- In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file.
- By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalog separately from the access programs. We call this property as **program-data** independence.
- The characteristic that allows program-data independence and program-operation independence is called **data abstraction**. A DBMS provides users with a conceptual representation of data that does not include many of the details of how the data is stored or how the operations are implemented.
- Informally, a data model is a type of data abstraction that is used to provide this conceptual representation. The data model uses logical concepts, such as objects, their properties, and their interrelationships, that may be easier for most users to understand than computer storage concepts. Hence, the data model hides storage and implementation details that are not of interest to most database users.

Support of Multiple Views of the Data

- A database typically has many types of users, each of whom may require a different perspective or view of the database.
- A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.
- Some users may not need to be aware of whether the data they refer to is **stored** or **derived**. A multiuser DBMS whose users have a variety of distinct applications must provide facilities for defining multiple views.

Sharing of Data and Multiuser Transaction Processing

- A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. This is essential if data for multiple applications is to be **integrated and maintained** in a single database.
- The DBMS must include **concurrency control** software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.
- A fundamental role of multiuser DBMS software is to ensure that concurrent transactions operate correctly and efficiently.
- The concept of a transaction has become central to many database applications. A transaction is an executing program or process that includes one or more database accesses, such as reading or updating of database records. Each transaction is supposed to execute a logically correct database access if executed in its entirety without interference from other transactions.

People associated with Database system

- For a small personal database, one person typically defines, constructs, and manipulates the database, and there is no sharing.
- However, in large organizations, many people are involved in the design, use, and maintenance of a large database with hundreds or thousands of users.
- We refer people whose jobs involve the day-to-day use of a large database as **actors on the scene** and call other as **workers behind the scene**—those who work to maintain the database system environment but who are not actively interested in the database contents as part of their daily job.

Database Administrators

- In a database environment, the primary resource is the database itself, and the secondary resource is the DBMS and related software. Administering these resources is the responsibility of the **database administrator** (DBA).
- The DBA is responsible for authorizing **access to the database**, coordinating and monitoring its use, and acquiring software and hardware resources as needed.
- The DBA is accountable for problems such as security breaches and poor system response time. In large organizations, the DBA is assisted by a staff that carries out these functions.

Database Designers

- Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to **represent and store** this data.
- These tasks are mostly undertaken before the database is actually implemented and populated with data. It is the responsibility of database designers to communicate with all prospective database users in order to **understand** their **requirements** and to create a design that meets these requirements.
- Database designers typically interact with each potential group of users and develop views of the database that meet the data and processing requirements of these groups. Each view is then analyzed and integrated with the views of other user groups. The final database design must be capable of **supporting the requirements of all user groups**.

End Users

- End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.
- There are several categories of end users

    - Casual end users occasionally access the database, but they may need different information each time. They use a sophisticated database query interface to specify their requests and are typically middle- or high-level managers or other occasional browsers.
    - Naive or parametric end users make up a sizable portion of database end users. Their main job function revolves around constantly querying and updating the database, using standard types of queries and updates— called canned transactions—that have been carefully programmed and tested.

- Many of these tasks are now available as mobile apps for use with mobile devices. The tasks that such users perform are varied. A few examples are:
    - Bank customers and tellers check account balances and post withdrawals and deposits.
    - Reservation agents or customers for airlines, hotels, and car rental companies check availability for a given request and make reservations.
    - Employees at receiving stations for shipping companies enter package identifications via bar codes and descriptive information through buttons to update a central database of received and in-transit packages.
    - Social media users post and read items on social media Web sites.
- Sophisticated end users include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.
- Standalone users maintain personal databases by using ready-made program packages that provide easy-to-use menu-based or graphics-based interfaces. An example is the user of a financial software package that stores a variety of personal financial data.

System Analysts and Application Programmers (Software Engineers)

- System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements.
- Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions. Such analysts and programmers—commonly referred to as software developers or software engineers—should be familiar with the full range of capabilities provided by the DBMS to accomplish their tasks.

Workers behind the Scene

Background workers are not interested in the database content itself and they include the following categories

- DBMS system designers and implementers design and implement the DBMS modules and interfaces as a software package. A DBMS is a very complex software system that consists of many components, or modules, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security. The DBMS must interface with other system software, such as the operating system and compilers for various programming languages.
- Tool developers design and implement tools—the software packages that facilitate database modeling and design, database system design, and improved performance. Tools are optional packages that are often purchased separately. They include packages for database design, performance monitoring, natural language or graphical interfaces, prototyping, simulation, and test data generation. In many cases, independent software vendors develop and market these tools.
- Operators and maintenance personnel (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Data models

- Data abstraction generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data.
- A data model—a collection of concepts that can be used to **describe the structure** of a database—provides the necessary means to achieve this abstraction.
- By structure of a database, we mean the data types, relationships, and constraints that apply to the data. Most data models also include a set of basic operations for specifying retrievals and updates on the database.
- Data models also handles the dynamic aspect or behaviour of a database application.

Categories of Data Models

Data models categorize according to the types of concepts they use to describe the database structure.

Low-level or physical data models

- Low-level or physical data models provide concepts that describe the details of **how** data is stored on the computer storage media, typically magnetic disks.
- Concepts provided by physical data models are generally meant for computer specialists, not for end users.

High-level or conceptual data models

- High-level or conceptual data models provides concepts that are close to the way many users perceive data. Conceptual data models use concepts such as **entities, attributes, and relationships**.
- High-level or conceptual data models provides the way to **access** the data desired way
- An entity represents a real-world object or concept, such as an employee or a project from the miniworld that is described in the database. An attribute represents some property of interest that further describes an entity, such as the employee's name or salary. A relationship among two or more entities represents an association among the entities.

Representational (or implementation) data models

- Representational (or implementation) data models provides concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage.
- Representational (or implementation) data models provides the high level of abstraction.

Schemas, Instances, and Database State

- The **description of a database** is called the database schema, which is specified during database design and is not expected to change frequently.
- Through Data models we can define the schemas, most data models have certain conventions for displaying schemas as diagrams.
- A displayed schema is called a schema diagram which is as below and it displays the structure of each record type but not the actual instances of records.
- Each object in the schema—such as STUDENT or COURSE—a schema construct.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

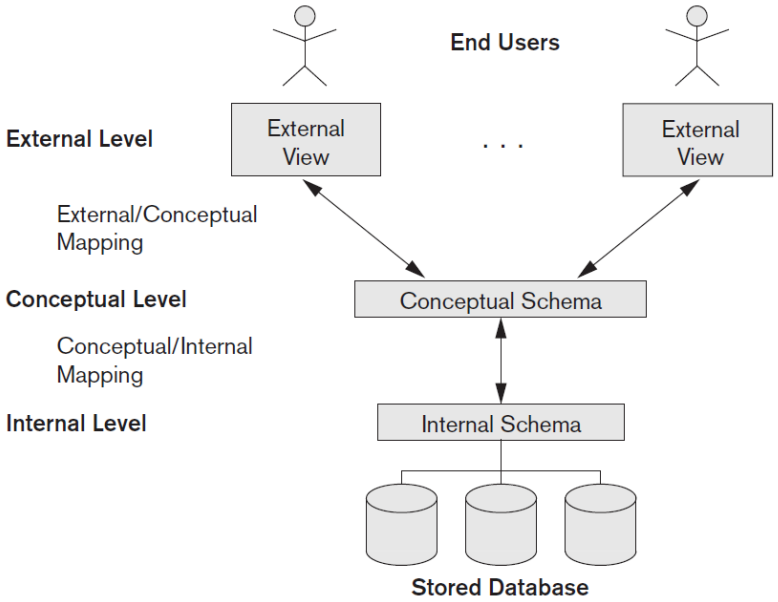
Student_number	Section_identifier	Grade

- **Database constraints** (conditions) are not showed in schema diagram, but the constraints are imposed during database operations.
- Usually database schema won't change frequently but data in database changes frequently.
- The data in the database at a particular moment in time is called a **database state or snapshot.** It is also called the current set of occurrences or instances in the database.
- Database states are classified as **initial state and current state**, we get the initial state of the database when the database is first populated or loaded, once database operations initiated the data starts to change and data at any point of time is current state of data.
- The DBMS is partly responsible for ensuring that every state of the database is a **valid state**, that is, a state that **satisfies the structure and constraints** specified in the schema.
- The DBMS stores the descriptions of the schema constructs and constraints also called the **meta-data**, in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to. The schema is sometimes called the **intension**, and a database state is called an **extension** of the schema.
- **Schema evolution** is the process of changing the schema which is not happen very often.

Database architecture

The Three-Schema Architecture

The goal of the three-schema architecture, illustrated in below figure, is to separate the user applications from the physical database.



In this architecture, schemas can be defined at the following three levels:

- The internal level has an internal schema, which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
- The conceptual level has a conceptual schema, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented. This implementation conceptual schema is often based on a conceptual schema design in a high-level data model.
- The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. As in the previous level, each external schema is typically implemented using a representational data model, possibly based on an external schema design in a high-level conceptual data model.

Note: The processes of transforming requests and results between levels are called mappings.

Data Independence

Data independence can be defined as the capacity to change the schema at one level of a database system without having to change the schema at the next higher level.

We can define two types of data independence

Logical data independence

- Logical data independence is the capacity to change the conceptual schema without having to change external schemas or application programs. We may change the conceptual schema to expand the database (by adding a record type or data item), to change constraints, or to reduce the database (by removing a record type or data item).
- In the last case, external schemas that refer only to the remaining data should not be affected.

Physical data independence

- Physical data independence is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well.
- Changes to the internal schema may be needed because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update. If the same data as before remains in the database, we should not have to change the conceptual schema.

Database Languages and Interfaces

The DBMS must provide appropriate languages and interfaces for each category of users. The languages are as fallows

- **Data definition language (DDL)**, is used by the DBA and by database designers to define both schemas. The DBMS will have a DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.
- **Storage definition language (SDL)**, is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages. In most relational DBMSs today, there is no specific language that performs the role of SDL. Instead, the internal schema is specified by a combination of functions, parameters, and specifications related to storage of files.
- **View definition language (VDL)**, to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas. In relational DBMSs, SQL is used in the role of VDL to define user or application views as results of predefined queries.
- **Data manipulation language (DML)** is the language which is used to operate on data, the operation may include retrieval, insertion, deletion, and modification of the data.
- There are two main types of DMLs. **A high-level or nonprocedural DML** can be

used on its own to specify complex database operations concisely. Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language.

- A **lowlevel or procedural DML** must be embedded in a general-purpose programming language.
- Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the **host language** and the DML is called the **data sublanguage**.On the other hand, a high-level DML used in a standalone interactive manner is called a **query language**.

DBMS Interfaces

User-friendly interfaces provided by a DBMS may include the following:

- **Menu-based Interfaces for Web Clients or Browsing.** These interfaces present the user with lists of options (called menus) that lead the user through the formulation of a request.

- **Apps for Mobile Devices**. These interfaces present mobile users with access to their data. For example, banking, reservations, and insurance companies, among many others, provide apps that allow users to access their data through a mobile phone or mobile device.

- **Forms-based Interfaces**. A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries. Forms are usually designed and programmed for naive users as interfaces to canned transactions.

- **Graphical User Interfaces.** A GUI typically displays a schema to the user in diagrammatic
form. The user then can specify a query by manipulating the diagram.
In many cases, GUIs utilize both menus and forms.

- **Natural Language Interfaces**. These interfaces accept requests written in English or some other language and attempt to understand them. A natural language interface usually has its own schema, which is similar to the database conceptual schema, as well as a dictionary of important words.

- **Keyword-based Database Search**. These are somewhat similar to Web search engines, which accept strings of natural language (like English or Spanish) words and match them with documents at specific sites (for local search engines) or Web pages on the Web at large (for engines like Google or Ask).

- **Speech Input and Output**. Limited use of speech as an input query and speech as an answer to a question or result of a request is becoming commonplace. Applications with limited vocabularies, such as inquiries for telephone directory, flight arrival/departure, and credit card account information, are allowing speech for input and output to enable customers to access this information.

- **Interfaces for Parametric Users.** Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a teller is able to use single function keys to invoke routine and repetitive transactions such as account deposits or withdrawals, or balance inquiries.

- **Interfaces for the DBA.** Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts, setting system parameters, granting account authorization, changing a schema, and reorganizing the storage structures of a database.

Classification of Database Management Systems

- Several criteria can be used to classify DBMSs. The first is the data model on which the DBMS is based. The main data model used in many current commercial DBMSs is the relational data model, and the systems based on this model are known as **SQL** systems.
- The object data model has been implemented in some commercial systems but has not had widespread use. Recently, so-called big data systems, also known as key-value storage systems and **NOSQL** systems, use various data models: document-based, graph-based, column-based, and key-value data models. Many legacy applications still run on database systems based on the hierarchical and network data models.
- The relational DBMSs are evolving continuously, and, in particular, have been incorporating many of the concepts that were developed in object databases. This has led to a new class of DBMSs called object-relational DBMSs. We can categorize
- DBMSs based on the data model: relational, object, object-relational, NOSQL, key-value, hierarchical, network, and other.
- Some experimental DBMSs are based on the XML (eXtended Markup Language) model, which is a tree-structured data model. These have been called native XML DBMSs. Several commercial relational DBMSs have added XML interfaces and storage to their products.
- The second criterion used to classify DBMSs is the number of users supported by the system. Single-user systems support only one user at a time and are mostly used with PCs. Multiuser systems, which include the majority of DBMSs, support concurrent multiple users.

E-R Model: Entity-Relationship modeling

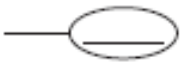
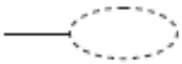
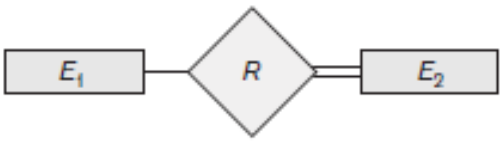
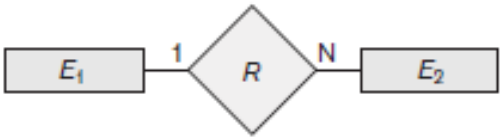
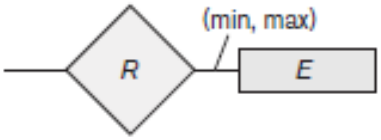
Introduction

Entity–relationship (ER) is modeling concept, which is a popular
high-level conceptual data model. This model and its variations are frequently used
for the conceptual design of database applications, and many database design tools
employ its concepts.
Diagrammatic notation associated with the ER model, known as ER diagrams.

Notation for ER diagrams

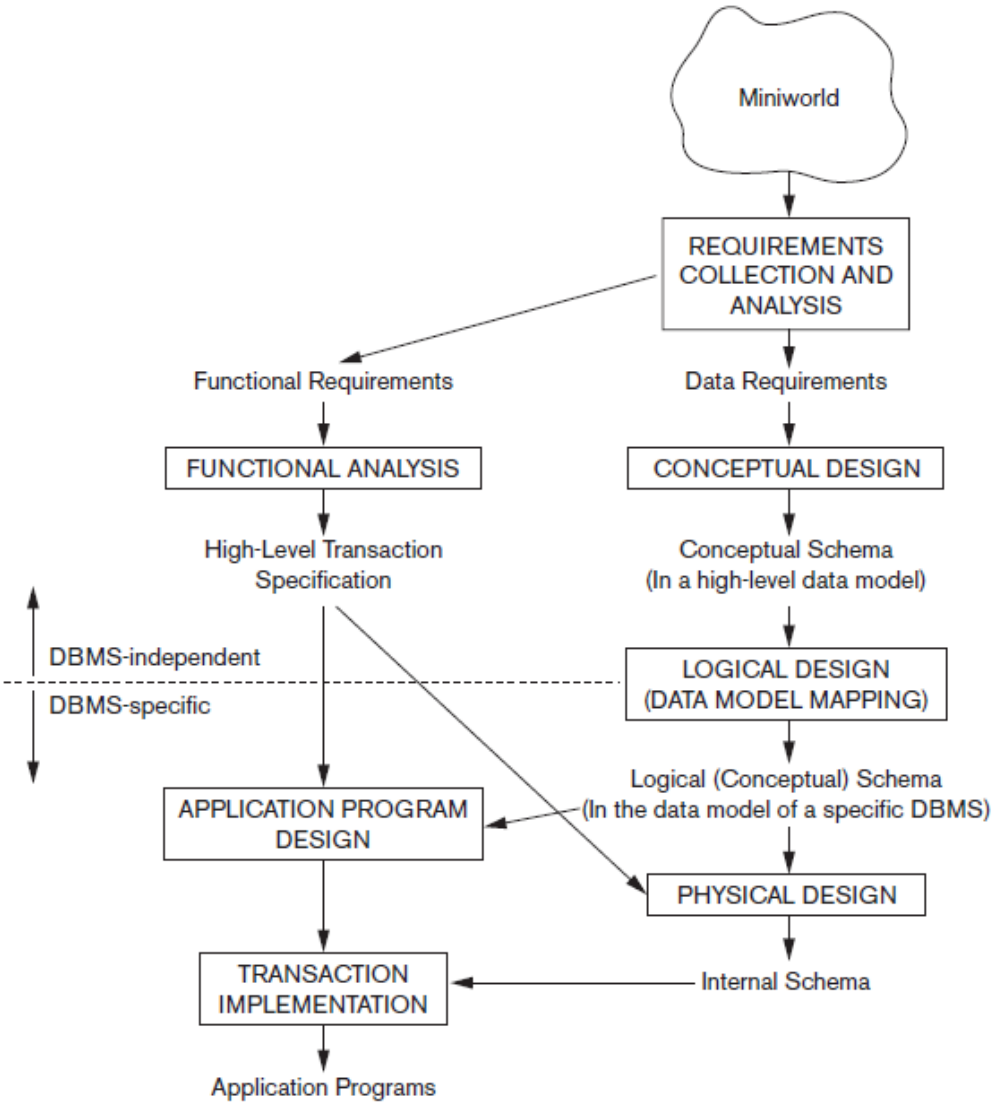
Note:

Below ER notations we are using throughout the note, prior knowledge of notations are necessary so we introducing the notation bit early, just a look at notations and synchronize the notations wherever required.

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Indentifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for $E_1 : E_2$ in R
	Structural Constraint (min, max) on Participation of E in R

Using High-Level Conceptual Data Models for Database Design

Below figure shows a simplified overview of the database design process.



- The first step shown is **requirements collection** and analysis. During this step, the database designers interview prospective database users to understand and document their data requirements.
- The result of this step is a concisely written set of users' requirements. These requirements should be specified in as detailed and complete a form as possible. In parallel with specifying the data requirements, it is useful to specify the known **functional requirements of the application**. These consist of the user defined operations (or transactions) that will be applied to the database, including both retrievals and updates.
- Once the requirements have been collected and analyzed, the next step is to create a

conceptual schema for the database, using a high-level conceptual data model and this step is called **conceptual design**. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints; these are expressed using the concepts provided by the high-level data model.

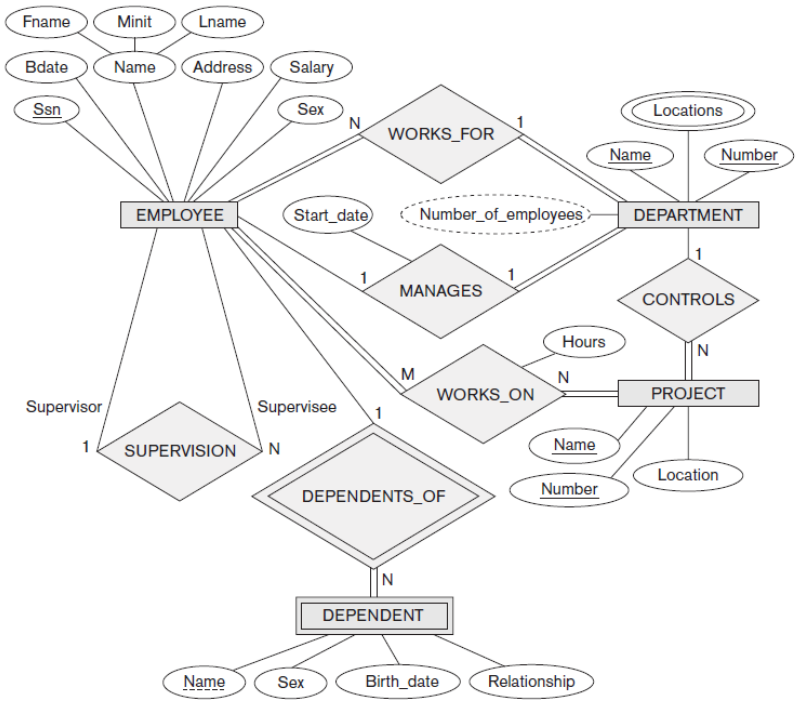
- During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user queries and operations identified during functional analysis.
- The next step is database design is the actual implementation of the database using a commercial DBMS and here conceptual schema is transformed from the high-level data model into the implementation data model. This step is called **logical design or data model mapping**;
- The last step is the **physical design phase**, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified.

A Sample Database Application

- In this sample database application called COMPANY, we make a walk through along various phases like requirement analysis, conceptual design, and logical design.
- Suppose that after the requirements collection and analysis phase, the database designers provide the following description of the miniworld—the part of the company that will be represented in the database as below.

    - The company is organized into departments. Each department has a unique name, a unique number, and a particular employee who manages the department. We keep track of the start date when that employee began managing the department. A department may have several locations.
    - A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
    - The database will store each employee's name, Social Security number,2 address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department. It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).
    - The database will keep track of the dependents of each employee for insurance purposes, including each dependent's first name, sex, birth date, and relationship to the employee.

- Below figure shows how the schema for this database application can be displayed by means of the graphical notation known as ER diagrams.



Entity Types, Entity Sets, Attributes, and Keys

- The ER model describes data as entities, relationships, and attributes.
- The entity-relationship (E-R) data model was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database.
- The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema.
  The E-R data model employs three basic concepts
  - Entity sets
  - Relationship sets
  - Attributes

Entity and Entity Sets

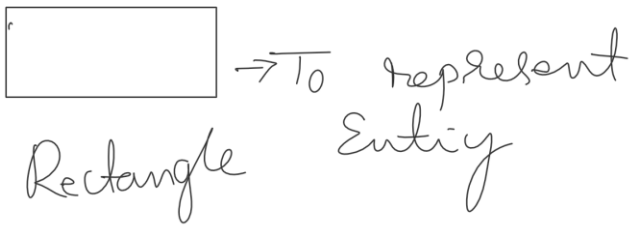
Entity

- An entity is a “thing” or “object” in the real world that is distinguishable from all other objects.

Or

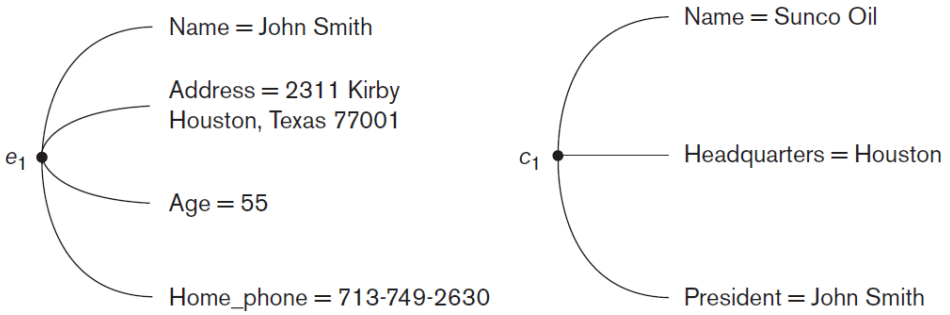
- An entity can be a real-world object, either animate or inanimate, that can be easily identifiable.

Note:
Rectangle shape is used to specify the entity set



Ex:

- Each person in a university is an entity. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity.
- In a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.
- Below figures shows two entities EMPLOYEE e1, and COMPANY c1, and their attributes.



Entity set and Entity Type

- An entity set is a set of entities of the same type that share the same properties, or attributes.
- Entity type defines a collection (or set) of entities that have the same attributes. Each Entity type in the database is described by its name and attributes.

Ex:

- The set of all people who are instructors at a given university can be defined as the entity set instructor.
- The entity set student might represent the set of all students in the university.
- Note: An entity is represented by a set of attributes. Attributes are descriptive properties possessed by each member of an entity set.

Below data can form the entity set of Instructors and Students

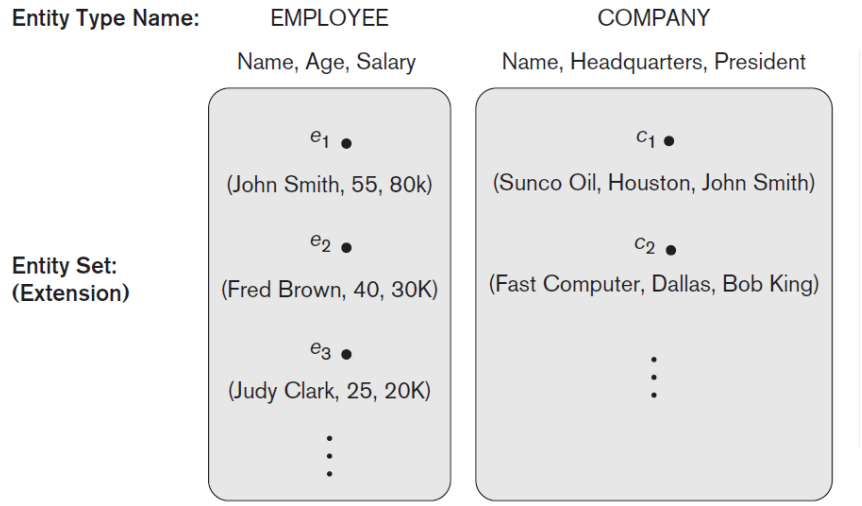
76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

98988	Tanaka
12345	Shankar
00128	Zhang
76543	Brown
76653	Aoi
23121	Chavez
44553	Peltier

student

Below data shows two entity types, EMPLOYEE and COMPANY, and some member entities of each.

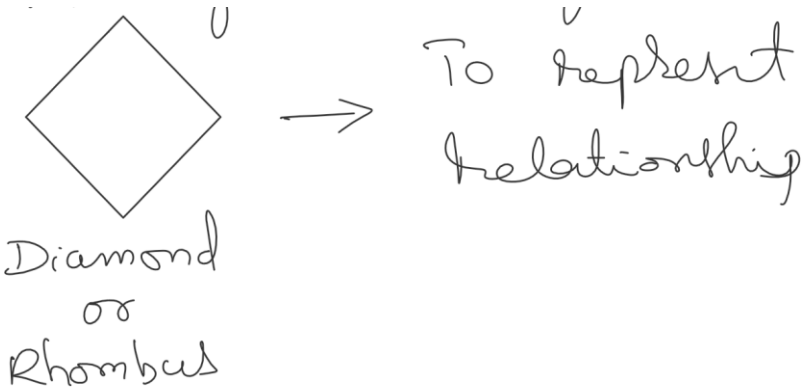


Relationship and Relationship Sets

Relationship

- A relationship is an association among several entities.
- Ex: We can define a relationship advisor that associates instructor Katz with student Shankar.
- This relationship specifies that Katz is an advisor to student Shankar.

Note:
Diamond Or Rhombus shape is used to specify the entity set



Relationship Sets

- A relationship set is a set of relationships of the same type. Formally, it is a mathematical relation on n ≥ 2 (possibly non distinct) entity sets.

  If E1, E2, . . . , En are entity sets, then a relationship set R is a subset of
  {(e1, e2, . . . , en) | e1 ∈ E1, e2 ∈ E2, . . . , en ∈ En}
  Where (e1, e2, . . . , en) is a relationship.

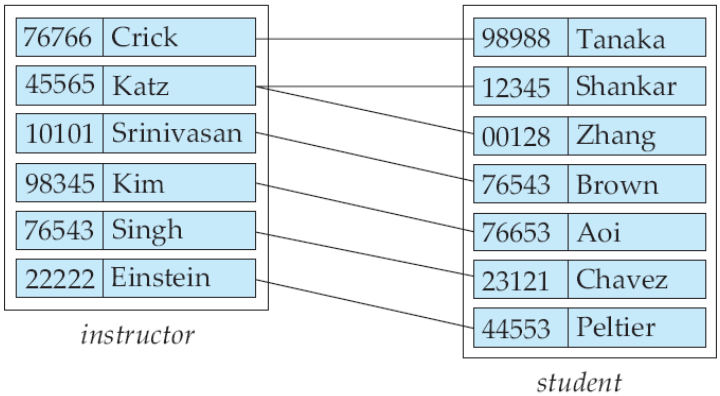
Participation

The association between entity sets is referred to as participation, that is the entity sets E1, E2, . . . , En participate in relationship set R.

Relationship instance

A relationship instance in an E-R schema represents an association between the named entities in the real-world enterprise that is being modelled.

Below data shows the example of relationship set.



Role
The function that an entity plays in a relationship is called that entity's role.

Recursive relationship set
The same entity set participates in a relationship set more than once, in different roles.

Descriptive attributes
A relationship may also have attributes called descriptive attributes.

Binary relationship set
More than one relationship set involving the same entity sets.

Degree of the relationship set

- The number of entity sets that participate in a relationship set is the **degree** of the relationship set.
- A binary relationship set is of degree 2
- A ternary relationship set is of degree 3.

Mapping Cardinalities

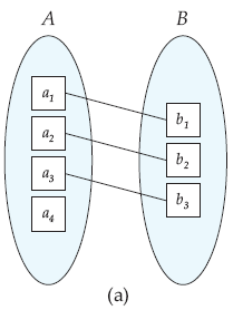
Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.

Mapping cardinalities are most useful in describing binary relationship sets, although they can contribute to the description of relationship sets that involve more than two entity sets.

For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following

One-to-one

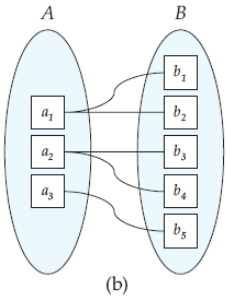
An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.



(a)

One-to-many

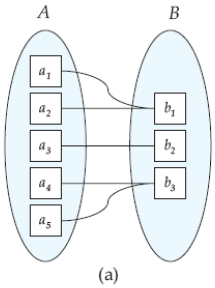
An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.



(b)

Many-to-one

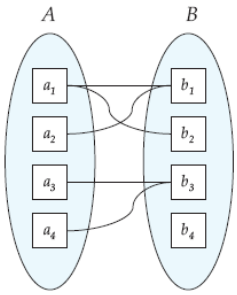
An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.



(a)

Many-to-many

An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated



(b)

Participation Constraints

Participation Constraints defines how the entity sets are participated in the relation, they are classified as below.

Total participation

The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R.

Partial participation

If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial.

Attributes

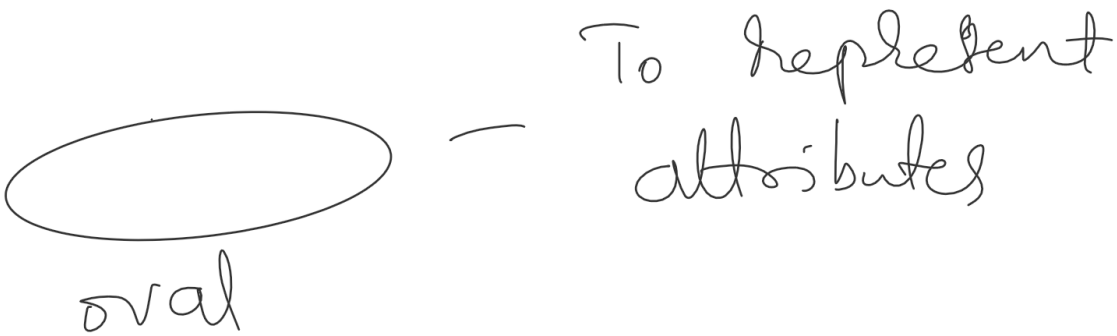
- Entities are represented by means of their properties, called attributes.

Or

- Set of properties of an entity is called attributes and attributes define the entity.
- All attributes have permitted values called domain or value set.

Ex:
A student entity may have name, class, and age as attributes.

Note:
Oval shape is used to specify the entity set



Types of Attributes

Simple attribute

- Simple attributes are atomic values, which cannot be divided further.
  Ex: A student's phone number is an atomic value of 10 digits.
- Adhar card no of Indian citizen

Composite attribute

- Composite attributes are made of more than one simple attribute.
  Ex: A student's complete name may have first_name and last_name.

Derived attribute

- Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database.
  Ex: average_salary in a department should not be saved directly in the database, instead it can be derived.

Age can be derived from data_of_birth.

Single-value attribute

- Single-value attributes contain single value.Ex: Social_Security_Number Or Passport Number.

Multi-value attribute

- Multi-value attributes may contain more than one values.
- Ex: A person can have more than one phone number, email_address, degree etc.

Note: An attribute takes a null value when an entity does not have a value for it.
The null value may indicate “not applicable”.

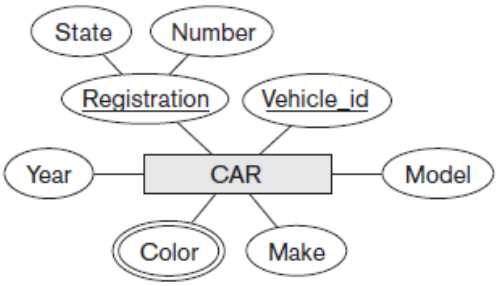
Key Attributes of an Entity Type

- Key attributes of an Entity Type is an important constraint on the entities of an entity type is the key or uniqueness constraint on attributes. An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely.
- For example, the Name attribute is a key of the COMPANY entity type in Figure 3.6 because no two companies are allowed to have the same name. For the PERSON entity type, a typical key attribute is Ssn (Social Security number).
- Sometimes several attributes together form a key, meaning that the combination of the attribute values must be distinct for each entity. If a set of attributes possesses this property, the proper way to represent this in the ER model that we describe here is to define a composite attribute and designate it as a key attribute of the entity type.
- Each key attribute has its name underlined inside the oval, while representing in ER Diagram.

Value Sets (Domains) of Attributes

Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.

Below diagram shows the Key attributes representation by considering the entity set CAR



CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

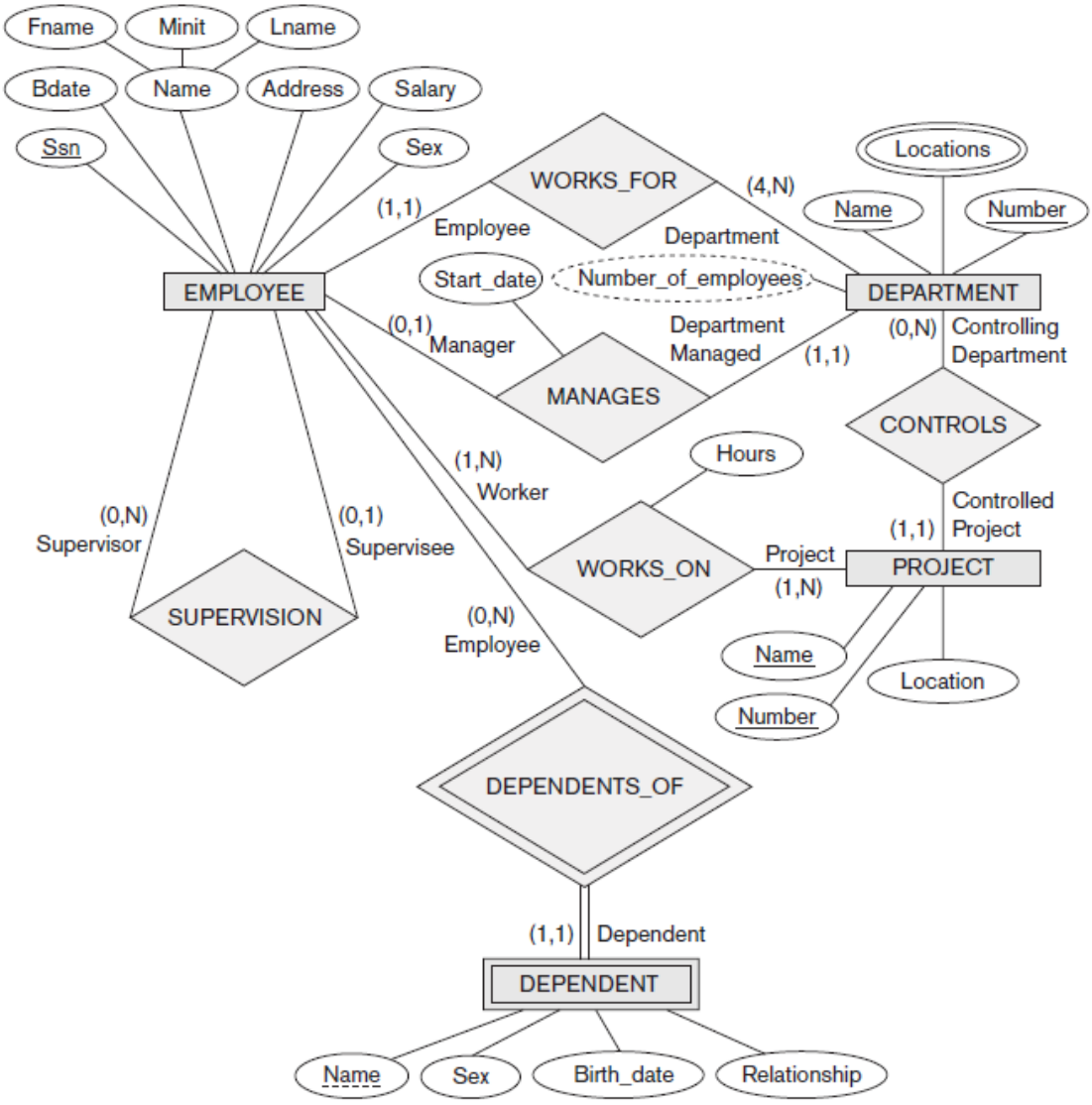
.
.
.

Weak Entity Types

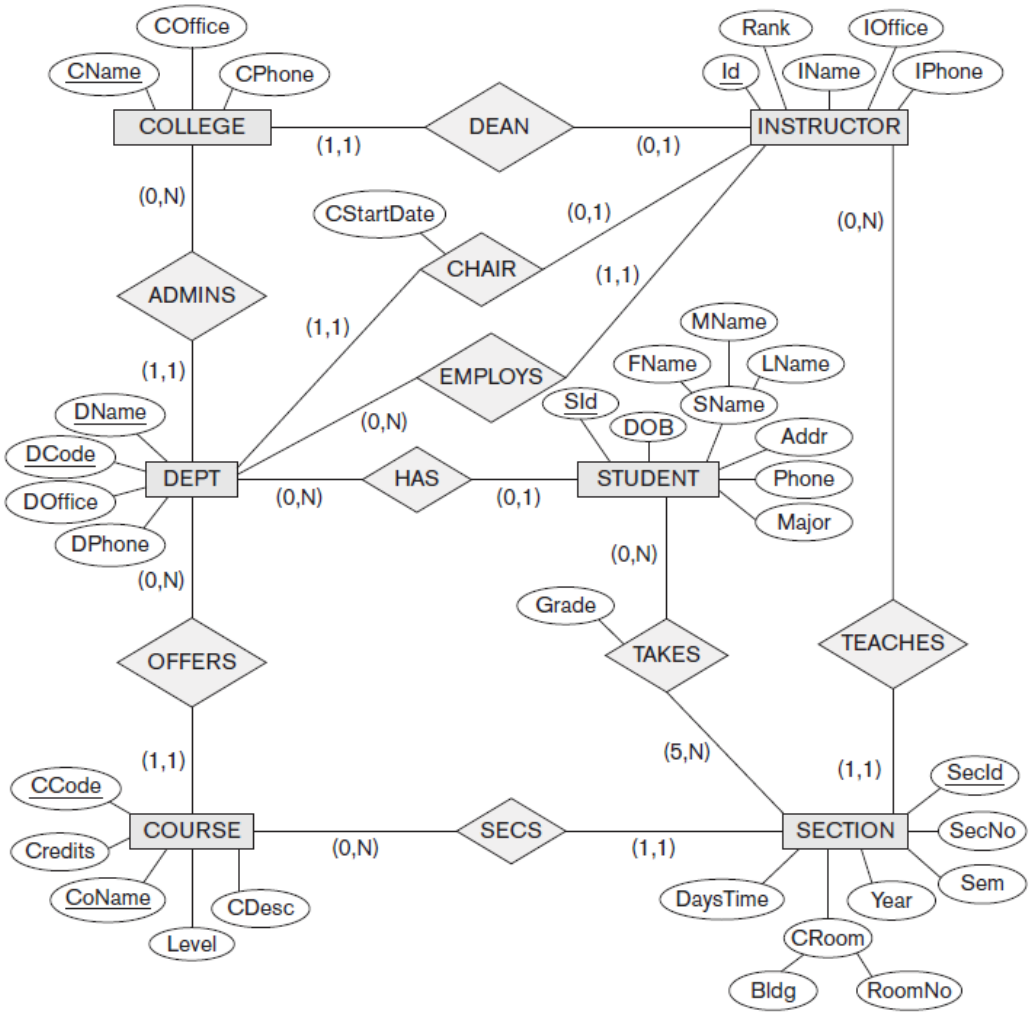
- Entity types that do not have key attributes of their own are called weak entity types
- Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the identifying or owner entity type, and we call the relationship type that relates a weak entity type to its owner the identifying relationship of the weak entity type.
- A weak entity type normally has a partial key, which is the attribute that can uniquely identify weak entities that are related to the same owner entity.

ER diagram example

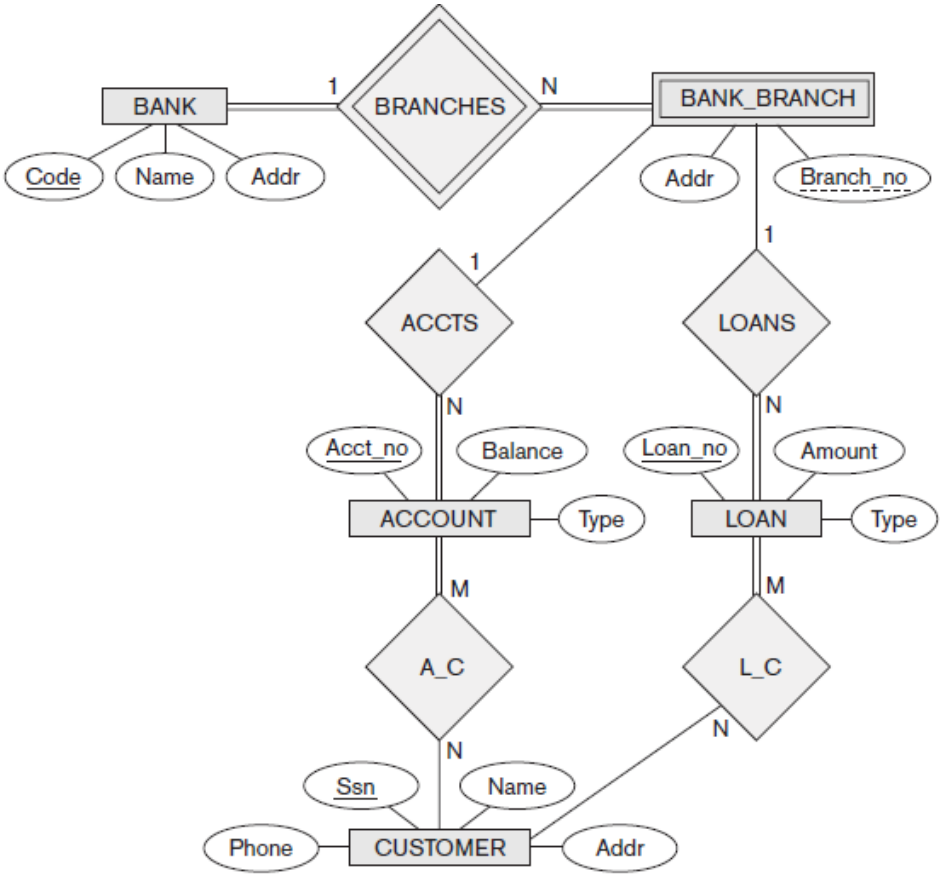
ER diagrams for the company schema.



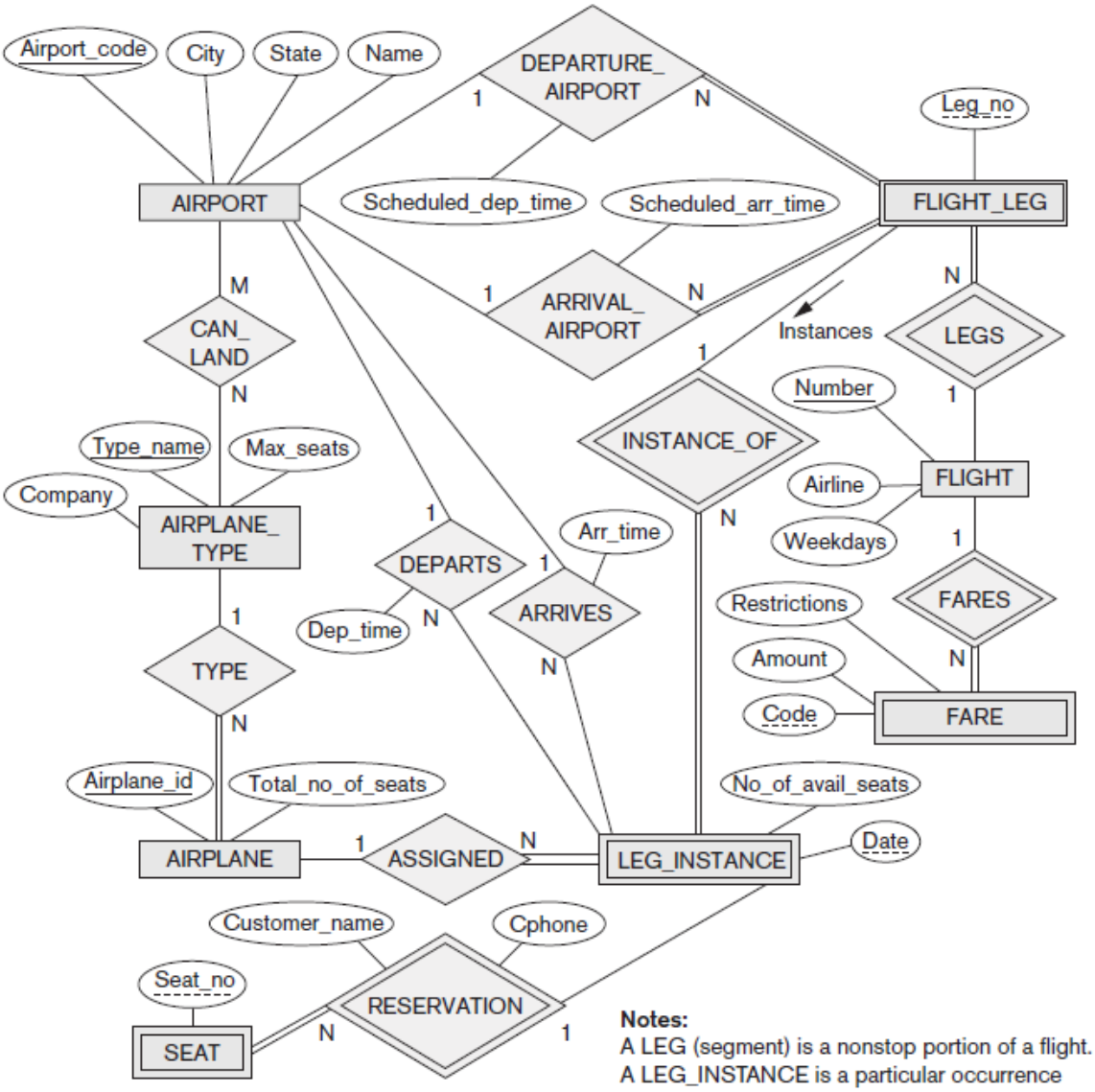
An ER diagram for a UNIVERSITY database schema.



An ER diagram for a BANK database schema



An ER diagram for an AIRLINE database schema.



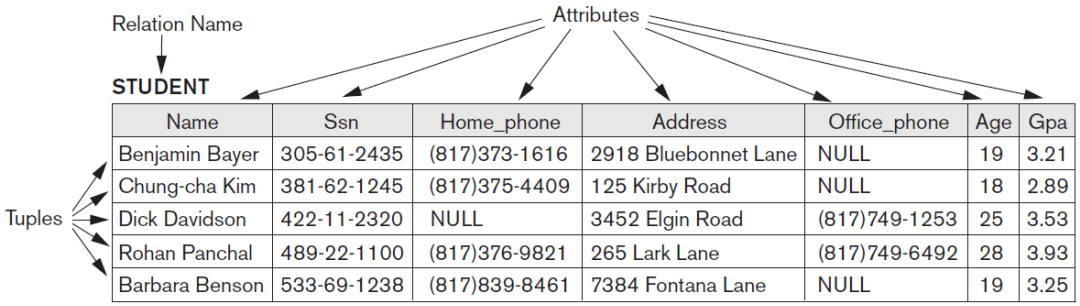
Unit II - Relational Data Model

Relational model concepts

"Relation nothing but table, relational database is all about storing the data in the form of table".

- The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values.
- When a relation is thought of as a table of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship.
- The table name and column names are used to help to interpret the meaning of the values in each row.

Below table shows the concepts of relational database like attributes, domain values, tuples, tables and so on…



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Basic concepts of relational database is as fallows

Relation or Tables

In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represent records and columns represent the attributes.

Tuple or row or record

A single row of a table, which contains a single record for that relation is called a tuple.

Relation instance

A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.

Relation schema

A relation schema describes the relation name (table name), attributes, and their names.

Relation key

Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely.

Attribute domain

Every attribute has some pre-defined value scope, known as attribute domain.

Atomic

A domain is atomic if elements of the domain are considered to be indivisible units.

Null value

The null value is a special value that signifies that the value is unknown or does not exist.

Characteristics of relations

- Ordering of tuples in a relation
    - Since a relation is a set of tuples and a set has no particular order among its elements, hence, tuples in a relation do not have any specified order. However, tuples in a relation can be logically ordered by the values of various attributes. In that case, information in a relation remains same, only the order of tuple varies. Hence, tuple ordering in a relation is irrelevant.
- Ordering of values within a tuple
    - An n-tuple is an ordered set of attribute values that belongs to the domain D, so, the order in which the values appear in the tuples is significant. However, if a tuple is defined as a set of (<attribute> : <value>) pairs, the order in which attributes appear in a tuple is irrelevant. This is due to the reason that there is no preference for one attribute value over another.

- Values and nulls in the tuples
    - Relational model is based on the assumption that each tuple in a relation contains a single value for each of its attribute. Hence, a relation does not allow composite and multivalued attributes. Moreover, it allows denoting the value of the attribute as null, if the value does not exist for that attribute or the value is unknown.
- No two tuples are identical in a relation
    - Since a relation is a set of tuples and a set does not have identical elements. Therefore, each tuple in a relation must be uniquely identified by its contents. In other words, two tuples with the same value for all the attributes (that is, duplicate tuples) cannot exist in a relation.

- Interpretation of a relation
  - A relation can be used to interpret facts about entities as a type of assertion. A relation can also be used to interpret facts about relationships.

Relational Model Constraints and Relational Database Schemas

While modeling the design of the relational database, we can put some restrictions or constraints or conditions like what values are allowed to be inserted in the relation, what kind of modifications and deletions are allowed in the relation.

Constraints in the databases can be categorized into 3 main categories:

- Constraints that are applied in the data model is called Implicit constraints.
- Constraints that are directly applied in the schemas of the data model, by specifying them in the DDL(Data Definition Language). These are called as schema-based constraints or Explicit constraints.
- Constraints that cannot be directly applied in the schemas of the data model. We call these Application based or semantic constraints.

Mainly Constraints on the relational database are of 4 types

- Domain constraints
- Key constraints
- Entity Integrity constraints
- Referential integrity constraints

Domain constraints

- Every domain must contain atomic values (smallest indivisible units) it means composite and multi-valued attributes are not allowed.
- We perform datatype check here, which means when we assign a data type to a column, we limit the values that it can contain.

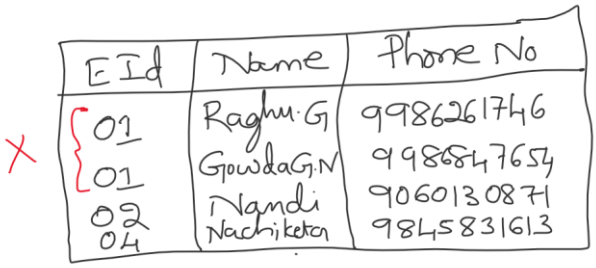
Eid	Name	PhoneNo
1	RaghuG	9986261746 9060130871

In the above relation, Name is a composite attribute and Phone is a multi-values attribute, so it is violating domain constraint.

Ex: If we assign the datatype of attribute age as int, we cant give it values other than int datatype.

Key Constraints or Uniqueness Constraints

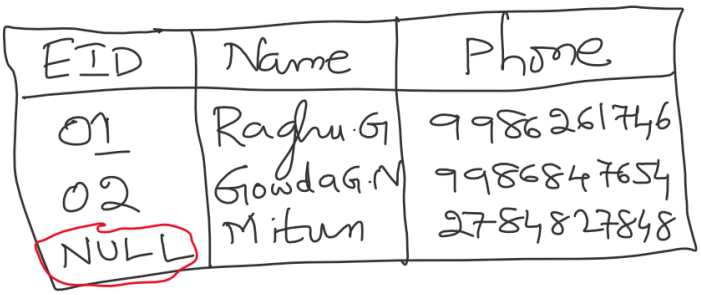
- These are called uniqueness constraints since it ensures that every tuple in the relation should be unique.
- A relation can have multiple keys or candidate keys (minimal super key), out of which we choose one of the keys as primary key, we don't have any restriction on choosing the primary key out of candidate keys, but it is suggested to go with the candidate key with less number of attributes.
- Null values are not allowed in the primary key, hence Not Null constraint is also a part of key constraint.



In the above table, EID is the primary key, and first and the second tuple has the same value in EID ie 01, so it is violating the key constraint.

Entity Integrity Constraints

Entity Integrity constraints says that no primary key can take NULL value, since using primary key we identify each tuple uniquely in a relation.



In the above relation, EID is made primary key, and the primary key cant take NULL values but in the third tuple, the primary key is null, so it is a violating Entity Integrity constraints.

Referential Integrity Constraints

The Referential integrity constraints is specified between two relations or tables and used to maintain the consistency among the tuples in two relations.
This constraint is enforced through foreign key, when an attribute in the foreign key of relation R1 have the same domain(s) as the primary key of relation R2, then the foreign key of R1 is said to reference or refer to the primary key of relation R2.
The values of the foreign key in a tuple of relation R1 can either take the values of the primary key for some tuple in relation R2, or can take NULL values, but can't be empty.

EId	Name	DID
01	Raghu G	01
02	Gowda G.N	02
03	Nandi	03

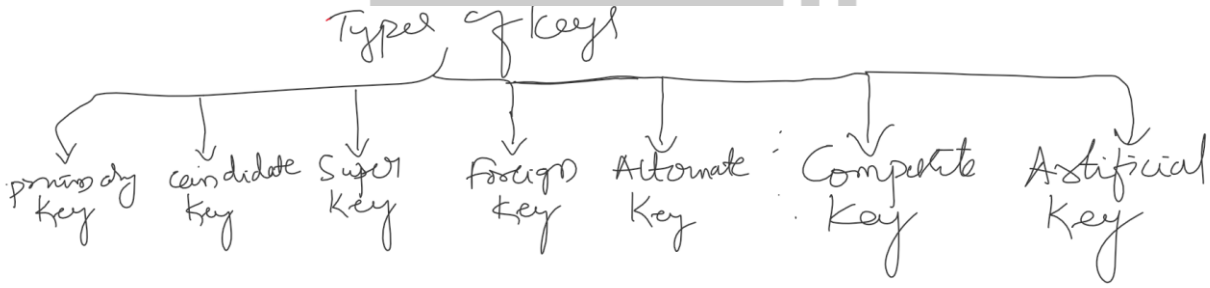
DID	DNAME
01	Management
02	Implementation

In the above tables, DID of the first relation is the foreign key, and DID in the second relation is the primary key. DID = 03 in the foreign key of the first table is not allowed since DID = 03 is not defined in the primary key of the second relation. Therefore, Referential integrity constraints is violated here.

Keys in Relational Database

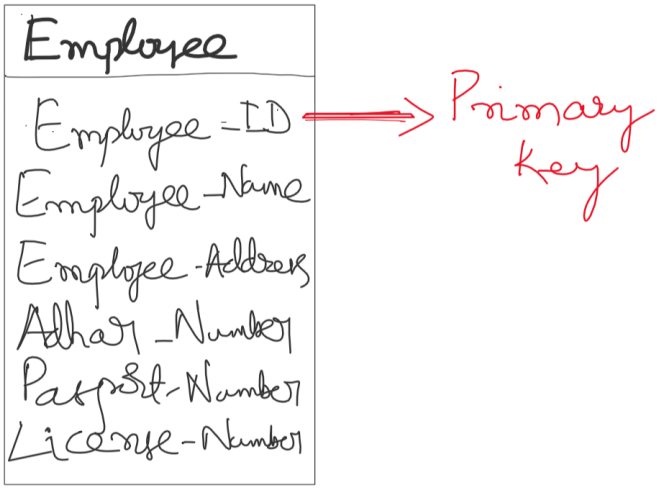
- Keys play an important role in the relational database.
- It is used to uniquely identify any record or row of data from the table. It is also used to establish and identify relationships between tables.

Types of Keys



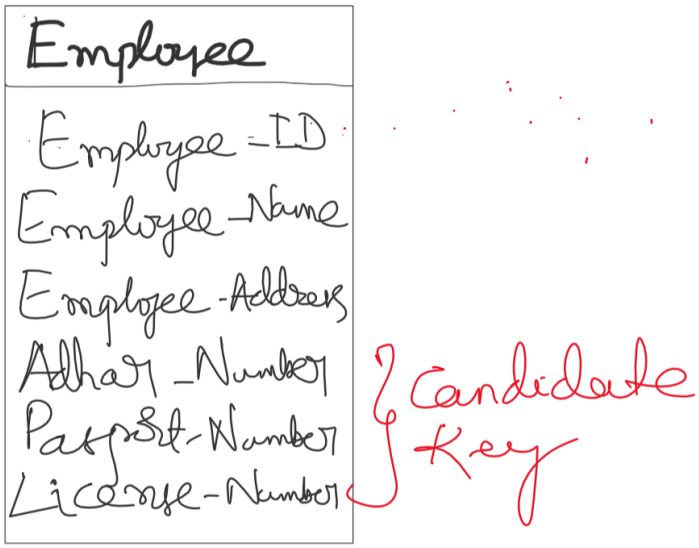
Primary key

- It is the first key used to identify one and only one instance of an entity uniquely. An entity can contain multiple keys, as we saw in the below EMPLOYEE table. The key which is most suitable from those lists becomes a primary key.
- In the EMPLOYEE table, ID can be the primary key since it is unique for each employee. In the EMPLOYEE table, we can even select License_Number and Adhar_Number as primary keys since they are also unique.
- For each entity, the primary key selection is based on requirements and developers.
- In below table Employee, Employee_ID is set as primary key and it should be unique and not NULL.



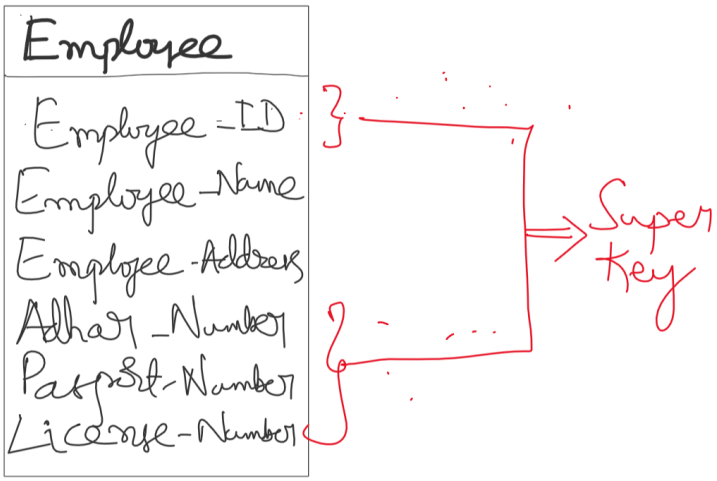
Candidate key

- A candidate key is an attribute or set of attributes that can uniquely identify a tuple.
- Except for the primary key, the remaining attributes are considered as candidate key. The candidate keys are as strong as the primary key.
- In below EMPLOYEE table, id is best suited for the primary key. The rest of the attributes, like Adhar_Number, Passport_Number, License_Number, etc., are considered a candidate key.



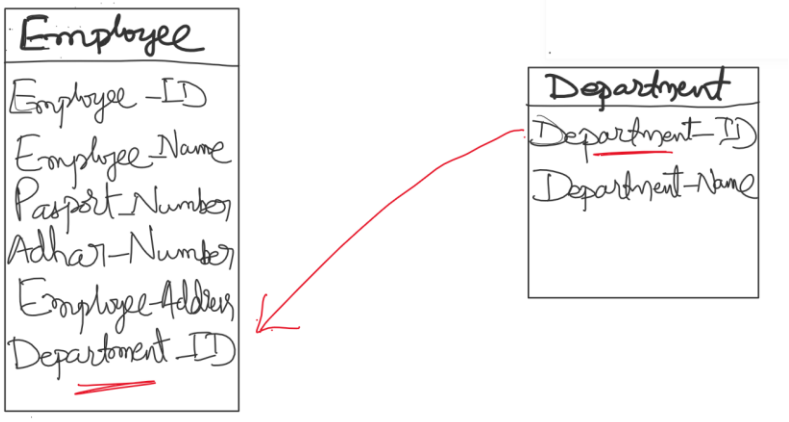
Super key

- Super key is an attribute set that can uniquely identify a tuple. A super key is a superset of a candidate key.
- In below table Employee columns or attributes like Employee_ID, Adhar_Number, Passport_Number, License_Number may used to identify each row or tuple uniquely, so this set of key are called as Super Key set, out of which may choose any one column as primary key and reaming elements belongs to group called candidate key.



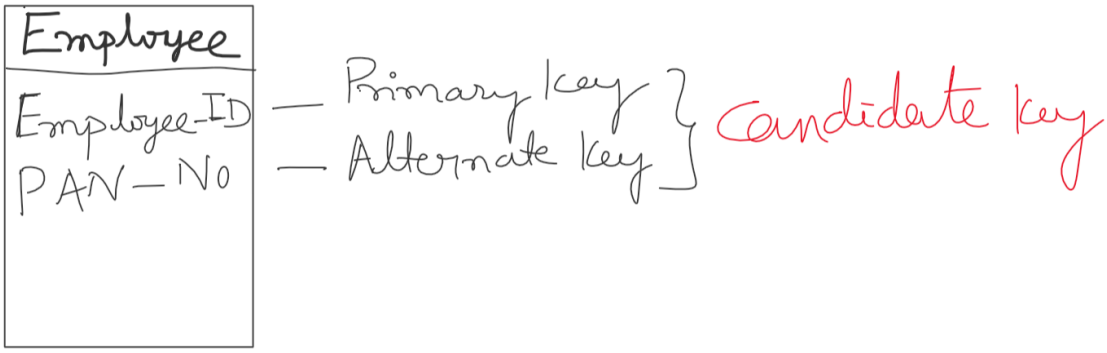
Foreign key

- Foreign keys are the column of the table used to point to the primary key of another table.
- Every employee works in a specific department in a company, and employee and department are two different entities. So we can't store the department's information in the employee table. That's why we link these two tables through the primary key of one table.
- We add the primary key of the DEPARTMENT table, Department_Id, as a new attribute in the EMPLOYEE table.
- In the EMPLOYEE table, Department_Id is the foreign key, and both the tables are related.



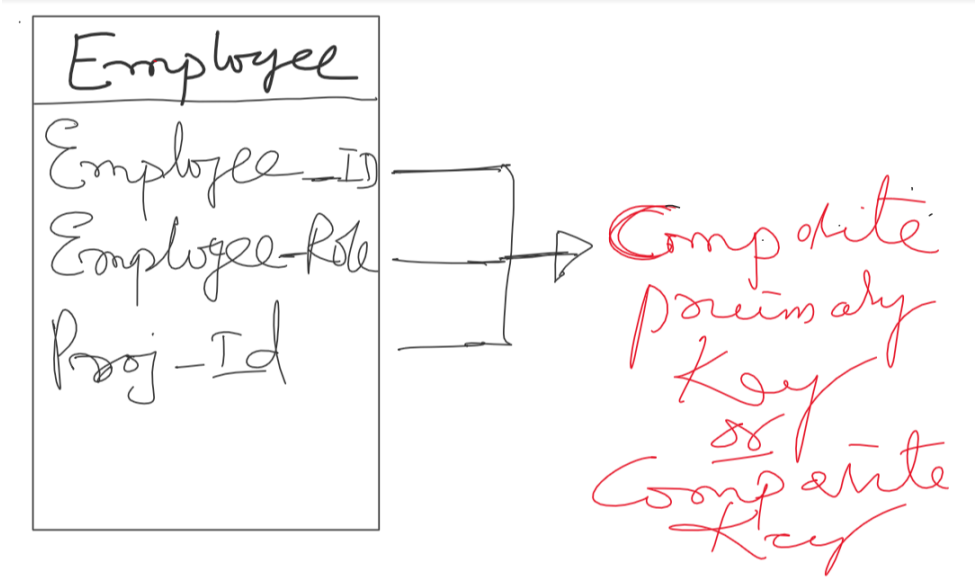
Alternate key

- There may be one or more attributes or a combination of attributes that uniquely identify each tuple in a relation. These attributes or combinations of the attributes are called the candidate keys. One key is chosen as the primary key from these candidate keys, and the remaining candidate key, if it exists, is termed the alternate key.
- In other words, the total number of the alternate keys is the total number of candidate keys minus the primary key. The alternate key may or may not exist. If there is only one candidate key in a relation, it does not have an alternate key.
- In the below employee relation has two attributes, Employee_Id and PAN_No, that act as candidate keys. In this relation, Employee_Id is chosen as the primary key, so the other candidate key, PAN_No, acts as the Alternate key.



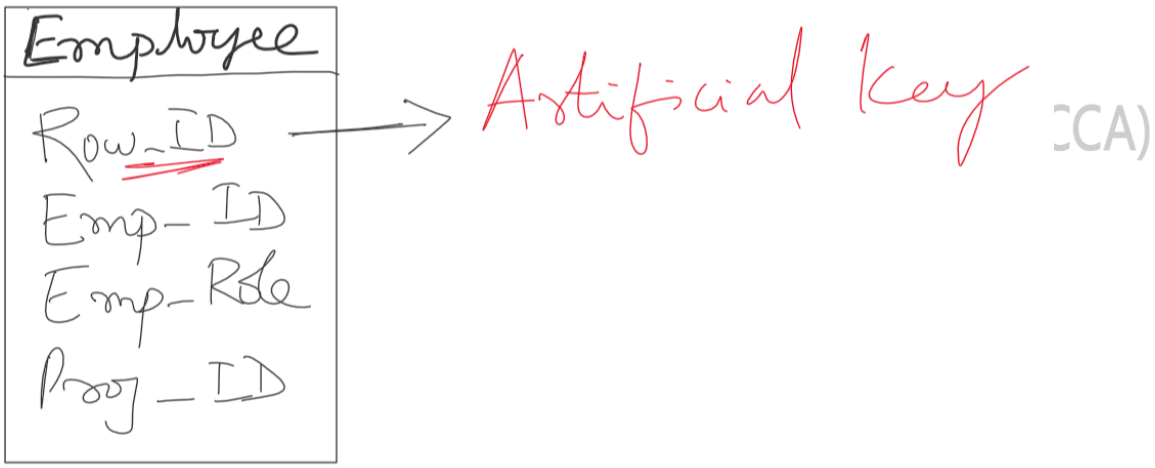
Composite key

- Whenever a primary key consists of more than one attribute, it is known as a composite key. This key is also known as Concatenated Key.
- Composite primary key is essential when a single attribute is not enough to fulfil the unique constrains and NULL constraints, in such cases we can combine more than one attributes and can produce composite key.
- In below employee relations, we assume that an employee may be assigned multiple roles, and an employee may work on multiple projects simultaneously. So the primary key will be composed of all three attributes, namely Emp_ID, Emp_role, and Proj_ID in combination. So these attributes act as a composite key since the primary key comprises more than one attribute.



Artificial key

- The key created using arbitrarily assigned data are known as artificial keys. These keys are created when a primary key is large and complex and has no relationship with many other relations. The data values of the artificial keys are usually numbered in a serial order.
- In above employee relation, the primary key, which is composed of Emp_ID, Emp_role, and Proj_ID, is large in employee relations. So it would be better to add a new virtual attribute to identify each tuple in the relation uniquely.



Relational Algebra

Basic Relational Algebra operations

Relational Algebra is procedural query language, which takes Relation as input and generate relation as output. Relational algebra mainly provides theoretical foundation for relational databases and SQL.

Basic Relational Algebra operations

Select Operation (σ)

- It selects tuples(rows) that satisfy the given predicate from a relation.
- The SELECT operation is used to choose a subset of the tuples from a relation that satisfies a selection condition. We can consider the SELECT operation to be a filter that keeps only those tuples that satisfy a qualifying condition.
- Alternatively, we can consider the SELECT operation to restrict the tuples in a relation to only those tuples that satisfy the condition. The SELECT operation can also be visualized as a horizontal partition of the relation into two sets of tuples—those tuples that satisfy the condition and are selected, and those tuples that do not satisfy the condition and are filtered out.
- Notation − σp(r)
- Where σ stands for selection predicate and r stands for relation. p is prepositional logic formula which may use connectors like and, or, and not. These terms may use relational operators like − =, ≠, ≥, < , > , ≤.
- In general, the SELECT operation is denoted by
  σ<selection condition>(R)

Ex:

σsubject = "database"(Books)
Output − Selects tuples from books where subject is 'database'.
SQL Version : select * from books where subject = ‘database’

σsubject = "database" and price = "450"(Books)
Output − Selects tuples from books where subject is 'database' and 'price' is 450.

σsubject = "database" and price = "450" or year > "2010"(Books)
Output − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

Project Operation (∏)

- It projects column(s) that satisfy a given predicate.
- SELECT operation chooses some of the rows from the table while discarding other rows. The PROJECT operation, on the other hand, selects certain columns from the table and discards the other columns.
- If we are interested in only certain attributes of a relation, we use the PROJECT operation to project the relation over these attributes only. Therefore, the result of the

PROJECT operation can be visualized as a vertical partition of the relation into two relations:

- Notation − ∏A1, A2, An (r)
- Where A1, A2 , An are attribute names of relation r.
- Duplicate rows are automatically eliminated, as relation is a set.
- The general form of the PROJECT operation is
  π<attribute list>(R)

Ex:

∏subject, author (Books)
Selects and projects columns named as subject and author from the relation Books.

Rename Operation (ρ)

- The results of relational algebra are also relations but without any name.
- The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter rho ρ.
- Notation − ρ x (E)
- Where the result of expression E is saved with name of x.
- The general RENAME operation when applied to a relation R of degree n is denoted by any of the following three forms
  ρS(B1, B2, ... , Bn)(R) or ρS(R) or ρ(B1, B2, ... , Bn)(R)
- Alternatively, we can explicitly show the sequence of operations, giving a name to each intermediate relation, and using the assignment operation, denoted by ← (left arrow), as follows:
  DEP5_EMPS ← σDno=5(EMPLOYEE)
  RESULT ← πFname, Lname, Salary(DEP5_EMPS)

Set theoretical operations on relations

- Several set theoretic operations are used to merge the elements of two sets in various ways, including UNION, INTERSECTION, and SET DIFFERENCE (also called MINUS or EXCEPT).
  These are binary operations; that is, each is applied to two sets (of tuples). When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples; this condition has been called union compatibility or type compatibility.
- Two relations R(A1, A2, … , An) and S(B1, B2, … , Bn) are said to be union compatible (or type compatible) if they have the same degree n and if dom(Ai) = dom(Bi) for 1 ≤ i ≤ n. This means that the two relations have the same number of attributes and each corresponding pair of attributes has the same domain.

Union Operation (∪)

- It performs binary union between two given relations and is defined as −
  r ∪ s = { t | t ∈ r or t ∈ s}
- Notation : r U s
- Where r and s are either database relations or relation result set (temporary relation).
- For a union operation to be valid, the following conditions must hold −
  - r, and s must have the same number of attributes.
  - Attribute domains must be compatible.
  - Duplicate tuples are automatically eliminated.

Ex:
∏ author (Books) ∪ ∏ author (Articles)

Output − Projects the names of the authors who have either written a book or an article or both.

Intersection Operation (∩)

- It displays the common values in R1 & R2. It is denoted by ∩.
- Syntax
  ∏regno(R1) ∩ ∏regno(R2)
- Consider two sets,
  A={1,2,4,6} and B={1,2,7}
  Intersection of A and B
  A ∩ B ={1,2}
- Elements that are present in both sets A and B are present in the set obtained by intersection of A and B.
- In relational algebra if R1 and R2 are two instances of relation then,
- R1 ∩ R2 ={ x | x€ R1 and x € R2}
- That is, the intersection of R1 and R2 only those tuples will be present that are in both R1 and R2

Set Difference (−)

- The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

- Notation : r − s
- Finds all the tuples that are present in r but not in s.

Ex:

- ∏ author (Books) − ∏ author (Articles)
- Output − Provides the name of authors who have written books but not articles.

- Cartesian product (X)

  - Combines information of two different relations into one.
  - Notation : r X s
  - Where r and s are relations and their output will be defined as :
  - r X s = { q t | q ∈ r and t ∈ s}

  Ex:
  σauthor = 'raghu'(Books X Articles)
  Output − Yields a relation, which shows all the books and articles written by raghu.

Extended Relational-Algebra Operations

Operations that cannot be expressed using the basic relational-algebra operations needs to be with special operations, they are called as extended relational-algebra operations.

Generalized Projection

- The first operation is the generalized-projection operation, which extends the projection operation by allowing operations such as arithmetic and string functions to be used in the projection list.

  The generalized-projection operation has the form:
  πF1,F2,...,Fn (E)

  Where

  E is any relational-algebra expression
  Each of F1, F2, . . . , Fn is an arithmetic expression involving constants and attributes in the schema of E.

  As a base case, the expression may be simply an attribute or a constant. In general, an expression can use arithmetic operations such as +,−, ∗, and ÷ on numeric valued attributes, numeric constants, and on expressions that generate a numeric result.

  Generalized projection also permits operations on other data types, such as concatenation of strings.

  Ex:
  π ID,name,dept name,salary÷12(instructor)

  Gives the ID, name, dept name, and the monthly salary of each instructor.

Aggregation

- The second extended relational-algebra operation is the aggregate operation G, which permits the use of aggregate functions such as min or average, on sets of values.
- Aggregate functions take a collection of values and return a single value as a result.

  Ex: The aggregate function sum takes a collection of values and returns the sum of the values. Thus, the function sum applied on the collection: {1, 1, 3, 4, 4, 11} returns the value 24.

  By considering instructor relation; suppose that we want to find out the sum of salaries of all instructors;
  The relational-algebra expression for this query is:
  $\mathcal{G}$sum(salary)(instructor)

  The symbol *G* is the letter G in calligraphic font; read it as "calligraphic G." The relational-algebra operation *G* signifies that aggregation is to be applied.

JOIN operations

Join is a combination of a Cartesian product followed by a selection process. A Join operation pairs two tuples from different relations, if and only if a given join condition is satisfied.

Theta (θ) Join

- Theta join combines tuples from different relations provided they satisfy the theta condition. The join condition is denoted by the symbol θ.
- Notation
  R1 ⋈θ R2
- R1 and R2 are relations having attributes (A1, A2, .., An) and (B1, B2,.. ,Bn) such that the attributes don't have anything in common, that is R1 ∩ R2 = Φ.
- Theta join can use all kinds of comparison operators.

Student		
SID	**Name**	**Std**
101	Alex	10
102	Maria	11

Subjects	
Class	**Subject**
10	Math
10	English
11	Music
11	Sports

Student_Detail –

$$STUDENT \bowtie_{Student.Std = Subject.Class} SUBJECT$$

Student_detail				
SID	**Name**	**Std**	**Class**	**Subject**
101	Alex	10	10	Math
101	Alex	10	10	English
102	Maria	11	11	Music
102	Maria	11	11	Sports

Equijoin

When Theta join uses only equality comparison operator, it is said to be equijoin. The above example corresponds to equijoin.

Natural Join (⋈)

- Natural join does not use any comparison operator. It does not concatenate the way a Cartesian product does. We can perform a Natural Join only if there is at least one common attribute that exists between two relations. In addition, the attributes must have the same name and domain.
- Natural join acts on those matching attributes where the values of attributes in both the relations are same.

Courses		
CID	**Course**	**Dept**
CS01	Database	CS
ME01	Mechanics	ME
EE01	Electronics	EE

HoD	
Dept	**Head**
CS	Alex
ME	Maya
EE	Mira

Courses ⋈ HoD			
Dept	**CID**	**Course**	**Head**
CS	CS01	Database	Alex
ME	ME01	Mechanics	Maya
EE	EE01	Electronics	Mira

Outer Joins

- Theta Join, Equijoin, and Natural Join are called inner joins. An inner join includes only those tuples with matching attributes and the rest are discarded in the resulting relation.
- Therefore, we need to use outer joins to include all the tuples from the participating relations in the resulting relation. There are three kinds of outer joins − left outer join, right outer join, and full outer join.

Left Outer Join(R ⟕ S)

- All the tuples from the Left relation, R, are included in the resulting relation. If there are tuples in R without any matching tuple in the Right relation S, then the S-attributes of the resulting relation are made NULL.

Left	
A	B
100	Database
101	Mechanics
102	Electronics

Right	
A	B
100	Alex
102	Maya
104	Mira

Courses ⟕ HoD			
A	B	C	D
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya

Right Outer Join: (R ⟖ S)

All the tuples from the Right relation, S, are included in the resulting relation. If there are tuples in S without any matching tuple in R, then the R-attributes of resulting relation are made NULL.

Courses ⟖ HoD			
A	B	C	D
100	Database	100	Alex
102	Electronics	102	Maya
---	---	104	Mira

Full Outer Join: (R ⟗ S)

All the tuples from both participating relations are included in the resulting relation. If there are no matching tuples for both relations, their respective unmatched attributes are made NULL.

Courses ⟗ HoD			
A	**B**	**C**	**D**
100	Database	100	Alex
101	Mechanics	---	---
102	Electronics	102	Maya
---	---	104	Mira

Aggregate Functions and Grouping

- Another type of request that cannot be expressed in the basic relational algebra is to specify mathematical aggregate functions on collections of values from the database.
- Common functions applied to collections of numeric values include SUM, AVERAGE, MAXIMUM, and MINIMUM. The COUNT function is used for counting tuples or values.
- We can define an AGGREGATE FUNCTION operation, using the symbol I (pronounced script F)7, to specify these types of requests as follows
  <grouping attributes> ℑ <function list> (R)
  where <grouping attributes> is a list of attributes of the relation specified in *R*, and <function list> is a list of (<function> <attribute>) pairs. In each such pair, <function> is one of the allowed functions—such as SUM, AVERAGE, MAXIMUM, MINIMUM, COUNT—and <attribute> is an attribute of the relation specified by *R*. The resulting relation has the grouping attributes plus one attribute for each element in the function list.

Ex:

To retrieve each department number, the number of employees in the department, and their average salary, while renaming the resulting attributes as indicated below, we write
ρR(Dno, No_of_employees, Average_sal) (Dno ℑ COUNT Ssn, AVERAGE Salary (EMPLOYEE))

Nested Sub Queries

- A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause.
- A subquery is used to return data that will be used in the main query as a condition to further restrict the data to be retrieved.
- Subqueries can be used with the SELECT, INSERT, UPDATE, and DELETE statements along with the operators like =, <, >, >=, <=, IN, BETWEEN, etc.

There are a few rules that subqueries must follow

- Subqueries must be enclosed within parentheses.
- A subquery can have only one column in the SELECT clause, unless multiple columns are in the main query for the subquery to compare its selected columns.
- An ORDER BY command cannot be used in a subquery, although the main query can use an ORDER BY. The GROUP BY command can be used to perform the same function as the ORDER BY in a subquery.
- Subqueries that return more than one row can only be used with multiple value operators such as the IN operator.
- A subquery cannot be immediately enclosed in a set function.
- The BETWEEN operator cannot be used with a subquery. However, the BETWEEN operator can be used within the subquery.
- A common use of subqueries are as below
  - Perform tests for set membership
  - Make set comparisons
  - Determine set cardinality

Note:

Subquery can be used in “where” as well as “from” clause.
We represent queries through relational algebra by using relational algebra signs, as below.

πF.nameσcnt<5γcount(E.snum) → cnt,F.*σC.name = E.cname ∧C.fid = F.fid[ρC(Class)×ρE(Enrolled)⋈RC.fid = F.fidρF(Faculty)]

Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table.
- The fields in a view are fields from one or more real tables in the database.
- We can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

  Syntax :

  CREATE VIEW view_name ASSELECT column1, column2, ...
  FROM table_name
  WHERE condition;

Note:
A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.
While creating views we use the same syntax as above, but the the queries are replaced by relational algebra queries.

SQL

Note

Dear Students,

The SQL portion what we mentioned below is not explicitly mentioned in syllabus, so when I preparing this notes, still some sort of ambiguity with respect to the inclusion of SQL portion here, so I suggesting you all that, go with SQL portion, since it included in lab syllabus and PL/SQL too.

Happy learning,
Raghu Gurumurthy

Introduction to SQL

- There are a number of database query languages in use
- SQL is the most widely used query language.
- SQL language as a “query language”.
- It can define the structure of the data, modify data in the database, and specify security constraints.
- IBM developed the original version of SQL, originally called Sequel, as part of the System R project in the early 1970s.
- In 1986, the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) published an SQL standard, called SQL-86.

The SQL language has several parts

- Data-definition language (DDL). The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.
- Data-manipulation language (DML). The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- Integrity: The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.
- View definition. The SQL DDL includes commands for defining views.
- Transaction control. SQL includes commands for specifying the beginning and ending of transactions.
- Embedded SQL and dynamic SQL. Embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, and Java.

- Authorization. The SQL DDL includes commands for specifying access rights to relations and views.

SQL Data Definition

The set of relations in a database must be specified to the system by means of a data definition language (DDL).

The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:

- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.
- The security and authorization information for each relation.
- The physical storage structure of each relation on disk.

Basic Types

The SQL standard supports a variety of built-in types as below

- char(n): A fixed-length character string with user-specified length n. The full form, character, can be used instead.
- varchar(n): A variable-length character string with user-specified maximum length n. The full form, character varying, is equivalent.
- int: An integer (a finite subset of the integers that is machine dependent). The full form, integer, is equivalent.
- smallint: A small integer (a machine-dependent subset of the integer type).
- numeric(p, d): A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, numeric(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.
- real, double precision: Floating-point and double-precision floating-point numbers with machine-dependent precision.
- float(n): A floating-point number, with precision of at least n digits.

Basic Schema Definition

We define an SQL relation by using the create table command.
The following command creates a relation department in the database.

```
create table department
(dept_name varchar (20) ,
building varchar (15),
budget numeric (12,2),
primary key (dept_name));
```

The relation created above has three attributes
dept name, which is a character
string of maximum length 20
building, which is a character string of maximum length 15, and budget, which is a number with 12 digits in total, 2 of which are after the decimal point.

The create table command also specifies that the dept name attribute is the primary key of the department relation.

The general form of the create table command is

```
create table r
(A1 D1,
 A2 D2,
 . . . ,
 An Dn,
(integrity-constraint1),
. . . ,
(integrity-constraintk));
```

where
r is the name of the relation
Each Ai is the name of an attribute in the schema of relation r,
Di is the domain of attribute Ai; that is, Di specifies the type of attribute Ai along with optional constraints that restrict the set of allowed values for Ai .
The semicolon shown at the end of the create table statements,.
SQL supports a number of different integrity constraints. Few of them as below

- Primary key (Aj1 , Aj2, . . . , Ajm): The primary-key specification says that attributes Aj1 , Aj2, . . . , Ajm form the primary key for the relation. The primarykey attributes are required to be nonnull and unique.

- Foreign key (Ak1 , Ak2, . . . , Akn) references s: The foreign key specification says that the values of attributes (Ak1 , Ak2, . . . , Akn) for any tuple in the relation must correspond to values of the primary key attributes of some tuple in relation s.

- Not null: The not null constraint on an attribute specifies that the null value is not allowed for that attribute; in other words, the constraint excludes the null value from the domain of that attribute.

Drop Table

To remove a relation from an SQL database, we use the drop table command.
The drop table command deletes all information about the dropped relation from the database.

Syntax:
drop table r;

Ex:

drop table Employee;

Here
Employee is the table name.
Employee table will erased along with the data in it.

Delete Tuples or Records or Rows from the table

Syntax:

delete from r;

Ex:
delete from Employee

Here
Employee is the table name.
All the data from the Employee got deleted but structure remains same.

The delete retains relation r or table, but deletes all tuples in r.
The drop table deletes not only all tuples of r, but also the schema for r.

Alter Table

Alter table command to used to alter attributes to an existing relation, the operations are as follows

- Adding new column to the table.

  Syntax:
  Alter table r add A D;

  Ex: Alter table Employee add DOB Date;

  Here
  Employee is the name of an existing relation.
  DOB is the name of the attribute to be added.
  Date is the type of the added attribute.

- Dropping existing column from the table.

  Syntax:
  Alter table r Drop Column A;

  Ex: Alter table Employee Drop Column DOB;

  Here
  Employee is the name of an existing relation.
  DOB is the name of the attribute to be dropped.

- Renaming the column in the table.

  Syntax:
  Alter table r Rename Column OldColumn_name to New Column_name;

  Ex:
  Alter table SalGrade rename column LowSal To LowestSalary

  Here
  SalGrade is the name of an existing relation.
  LowSal is the name of the column to be rename.
  LowestSalary is the new name of the column

- Adding or Deleting constraints from the table.

  Syntax:
  alter table r add constraint cn cn_type(column);

  Ex:
  Alter table SalGrade add constraint salgrade_pk PRIMARY KEY(GRADE);

  Here
  SalGrade is the name of an existing relation.
  salgrade_pk is the name of the constraint.
  Primary key – Keyword
  Grade – Column needs to make primary key.

  Ex:

  Alter table Salgrade Drop constraint salgrage_pk;

Basic Structure of SQL Queries

The basic structure of an SQL query consists of three clauses

- select (What columns we need to select)
- from (From which table or relation we need select)
- where (Condition select respective rows)

The query takes as its input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then produces a relation as the result.

Queries can be made on single table or multiple tables

Single table queries

Consider the below table Employee for single table queries

EMPLOYEE

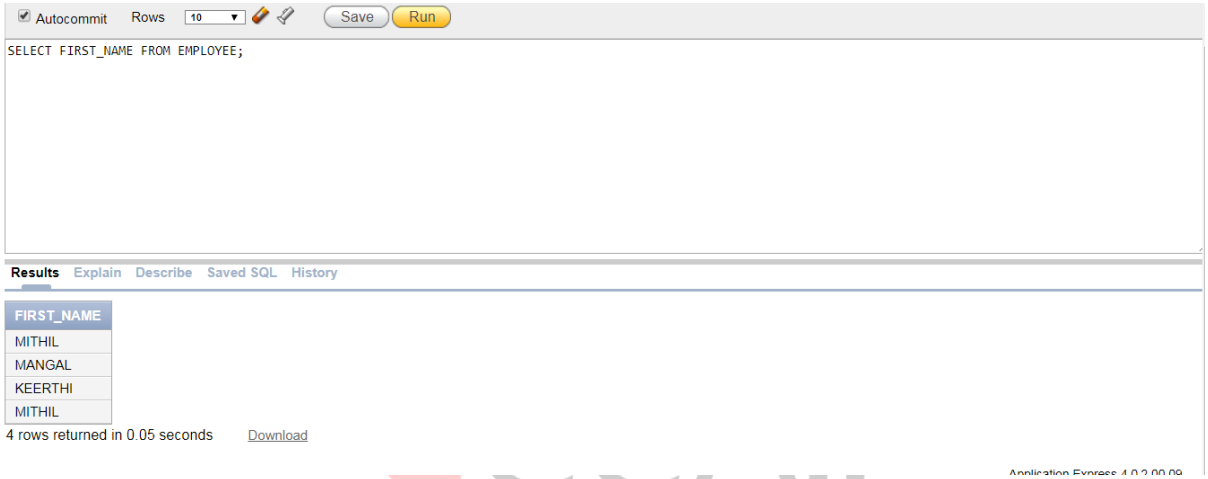
Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL

Query | Count Rows | Insert Row

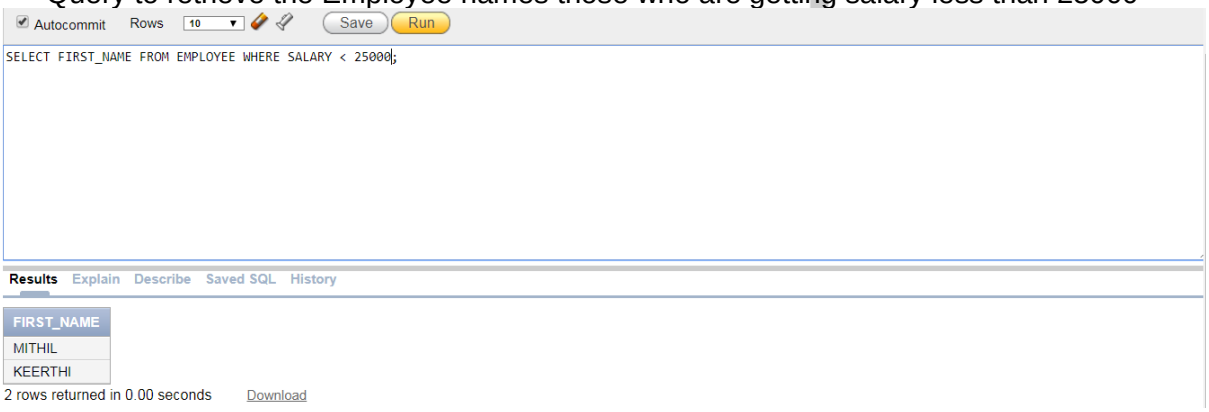
EDIT	EMP_ID	FIRST_NAME	LAST_NAME	JOB_ID	DOJ	SALARY	DEPT_ID	MANAGER_ID	DOB	PHONE
	1	MITHIL	SHANKAR	J1	01/01/2010	23000	1	1	01/23/1990	9845831613
	2	MANGAL	PANDEY	J2	02/01/2011	25000	2	2	10/23/1986	2784825878
	3	KEERTHI	PRASAD	J3	02/01/2011	20000	3	3	10/23/1986	2784825878
	4	MITHIL	SHANKAR	J3	01/01/2010	28000	1	3	10/23/1986	2784825878

row(s) 1 - 4 of 4

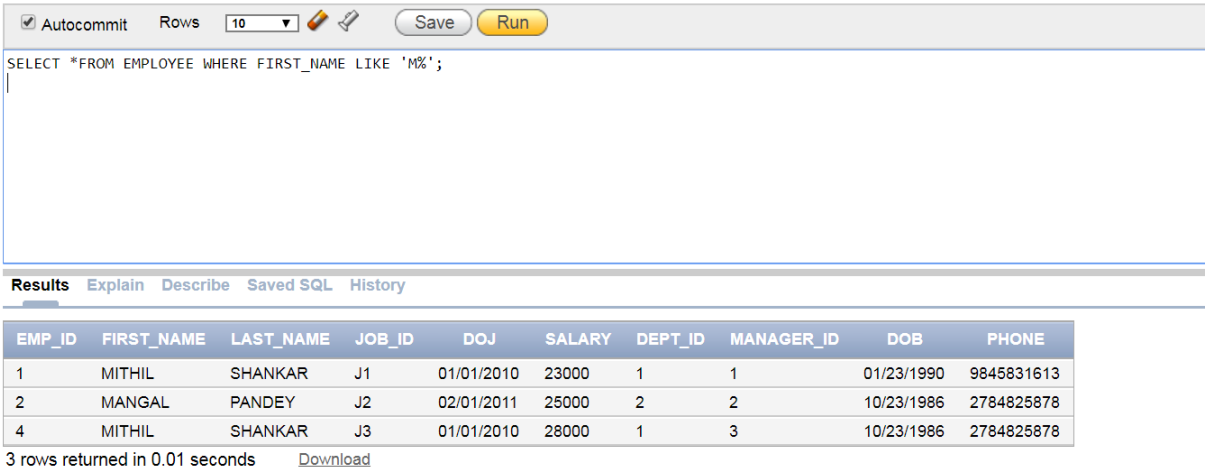
- Query to retrieve all first_name from table Employee.



- Query to retrieve the Employee names those who are getting salary less than 25000



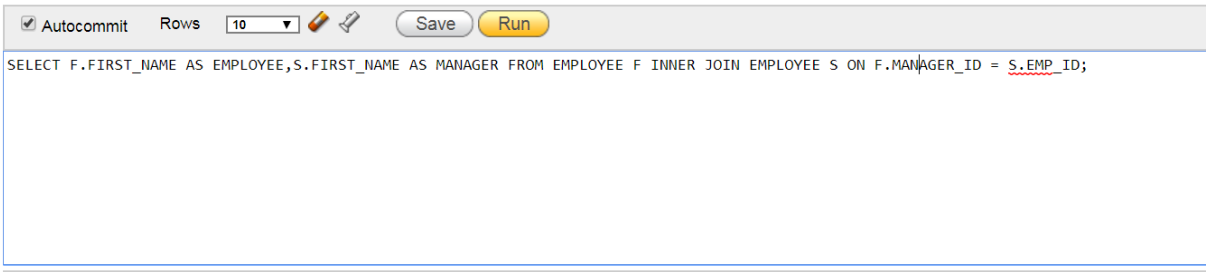
- Query to retrieve the Employee details whose name starts with M



EMP_ID	FIRST_NAME	LAST_NAME	JOB_ID	DOJ	SALARY	DEPT_ID	MANAGER_ID	DOB	PHONE
1	MITHIL	SHANKAR	J1	01/01/2010	23000	1	1	01/23/1990	9845831613
2	MANGAL	PANDEY	J2	02/01/2011	25000	2	2	10/23/1986	2784825878
4	MITHIL	SHANKAR	J3	01/01/2010	28000	1	3	10/23/1986	2784825878

3 rows returned in 0.01 seconds Download

- Query to retrieve the Employee name and their Managers name



EMPLOYEE	MANAGER
MITHIL	MITHIL
MANGAL	MANGAL
KEERTHI	KEERTHI
MITHIL	KEERTHI

4 rows returned in 0.07 seconds Download

Multiple table queries

The Natural Join

- The natural join operation operates on two relations and produces a relation as the result. Unlike the Cartesian product of two relations, which concatenates each tuple of the first relation with every tuple of the second.
- Natural join considers only those pairs of tuples with the same value on those attributes that appear in the schemas of both relations.

Consider below tables Employee and Departments

EMPLOYEE

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL

Query | Count Rows | Insert Row

EDIT	EMP_ID	FIRST_NAME	LAST_NAME	JOB_ID	DOJ	SALARY	DEPT_ID	MANAGER_ID	DOB	PHONE
	1	MITHIL	SHANKAR	J1	01/01/2010	23000	1	1	01/23/1990	9845831613
	2	MANGAL	PANDEY	J2	02/01/2011	25000	2	2	10/23/1986	2784825878
	3	KEERTHI	PRASAD	J3	02/01/2011	20000	3	3	10/23/1986	2784825878
	4	MITHIL	SHANKAR	J3	01/01/2010	28000	1	3	10/23/1986	2784825878

row(s) 1 - 4 of 4

DEPARTMENTS

Table | Data | Indexes | Model | Constraints | Grants | Statistics | UI Defaults | Triggers | Dependencies | SQL

Query | Count Rows | Insert Row

EDIT	DEPT_ID	DEPT_NAME	MANAGER_ID
	1	PRODUCTION	1
	2	SALES	2
	3	FINANCE	3
	4	TESTING	4

row(s) 1 - 4 of 4

Download

- Query to retrieve the Employee name and Department names from the Employee and Departments table.

Autocommit | Rows 10 | Save | Run

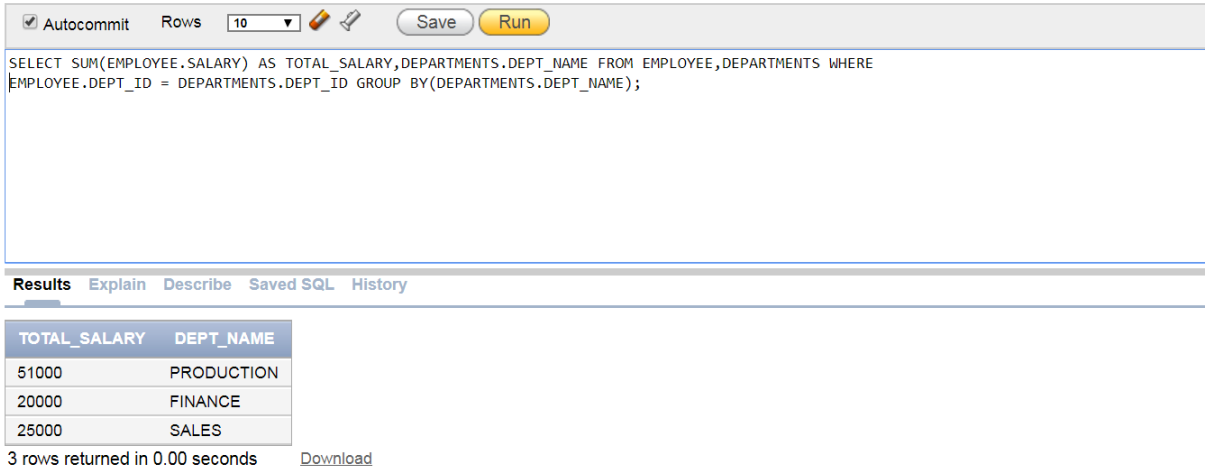
```
SELECT FIRST_NAME,DEPT_NAME FROM EMPLOYEE,DEPARTMENTS WHERE EMPLOYEE.DEPT_ID = DEPARTMENTS.DEPT_ID;
```

Results | Explain | Describe | Saved SQL | History

FIRST_NAME	DEPT_NAME
MITHIL	PRODUCTION
MITHIL	PRODUCTION
MANGAL	SALES
KEERTHI	FINANCE

4 rows returned in 0.00 seconds Download

- Query to retrieve the Department wise Total Salary By combining Employee and Departments table.



Additional Basic Operations

There are number of additional basic operations that are supported in SQL.

- The Rename Operation
  - SQL provides a way of renaming the attributes of a result relation.
  - It uses the as clause, taking the form:
    old-name as new-name
  - The as clause can appear in both the select and from clauses.

Ex:

```
select name as instructor name, course id
from instructor, teaches
where instructor.ID= teaches.ID;
```



```
select T.name, S.course id
from instructor as T, teaches as S
where T.ID= S.ID;
```

String Operations

- SQL specifies strings by enclosing them in single quotes.

  Ex: ‘Computer’

- A single quote character that is part of a string can be specified by using two single quote characters.

  Ex: The string “It’s right” can be specified by “It”s right”.

- The SQL standard specifies that the equality operation on strings is case sensitive as a result the expression "'comp. sci.' = 'Comp. Sci.'" evaluates to false.
- However, some database systems, such as MySQL and SQL Server, do not distinguish uppercase from lowercase when matching strings; as a result "'comp. sci.' = 'Comp. Sci.'" would evaluate to true on these databases.
- Pattern matching can be performed on strings, using the operator like. We describe patterns by using two special characters.

Percent (%): The % character matches any substring.
Underscore (_): The character matches any character.

Patterns are case sensitive; that is, uppercase characters do not match lowercase characters, or vice versa.

Ex:

- 'Intro%' matches any string beginning with "Intro".
- '%Comp%' matches any string containing "Comp" as a substring, for example, 'Intro. to Computer Science', and 'Computational Biology'.
- '___' matches any string of exactly three characters.(Uma,Ram)
- '___%' matches any string of at least three characters.(Uma,Ram,Ramu,Sinchana,Shrithi)

- SQL expresses patterns by using the like comparison operator.

Ex:
```
select dept_name
from department
where building like '%Watson%';
```

Return the names of all departments whose building name includes the substring 'Watson'.

For patterns to include the special pattern characters (that is,% and _), SQL allows the specification of an escape character.
The escape character is used immediately before a special pattern character to indicate that the special pattern character is to be treated like a normal character.

Ex:

- like 'ab\%cd%' escape '\' matches all strings beginning with "ab%cd".
- like 'ab\\cd%' escape '\' matches all strings beginning with "ab\cd".

Note: In oracle 11g the eacape sequence is % for % and for _ for _;

Attribute Specification in Select Clause

- The asterisk symbol " * " can be used in the select clause to denote "all attributes."
- Thus, the use of instructor.* in the select clause of the query

Ex:

```
select instructor.*
from instructor, teaches
where instructor.ID= teaches.ID;
```

Above query indicates that all attributes of instructor are to be selected.
A select clause of the form select * indicates that all attributes of the result relation of the from clause are selected.

Where Clause Predicates

SQL includes a between comparison operator to simplify where clauses that specify that a value be less than or equal to some value and greater than or equal to some other value.

Ex:
If we wish to find the names of employees with salary amounts between 23000 and 28000, we can use the between comparison as below.

```
Select * from employee where salary between 23000 and 28000;
```
Or
```
Select * from employee where salary >= 23000 and salary <= 28000;
```

Ordering the Display of Tuples

- SQL offers the user some control over the order in which tuples in a relation are displayed.
- The order by clause causes the tuples in the result of a query to appear in sorted order.

Ex:


```
SELECT * FROM EMPLOYEE WHERE SALARY >= 23000 ORDER BY LAST_NAME
DESC;
```

OUTPUT:

EMP_ID	FIRST_NAME	LAST_NAME	JOB_ID	DOJ	SALARY	DEPT_ID	MANAGER_ID	DOB	PHONE
1	MITHIL	SHANKAR	J1	01/01/2010	23000	1	1	01/23/1990	9845831613
4	MITHIL	SHANKAR	J3	01/01/2010	28000	1	3	10/23/1986	2784825878
2	MANGAL	PANDEY	J2	02/01/2011	25000	2	2	10/23/1986	2784825878

3 rows returned in 0.00 seconds Download

- By default, the order by clause lists items in ascending order.
- To specify the sort order, we may specify desc for descending order or asc for ascending order.
- Ordering can be performed on multiple attributes.

Ex:

SELECT * FROM EMPLOYEE WHERE SALARY >= 23000 ORDER BY LAST_NAME DESC,PHONE ASC;

OUTPUT:

EMP_ID	FIRST_NAME	LAST_NAME	JOB_ID	DOJ	SALARY	DEPT_ID	MANAGER_ID	DOB	PHONE
4	MITHIL	SHANKAR	J3	01/01/2010	28000	1	3	10/23/1986	2784825878
1	MITHIL	SHANKAR	J1	01/01/2010	23000	1	1	01/23/1990	9845831613
2	MANGAL	PANDEY	J2	02/01/2011	25000	2	2	10/23/1986	2784825878

3 rows returned in 0.01 seconds Download

Set Operations

- The SQL operations union, intersect, and except operate on relations and correspond to the mathematical set-theory operations ∪, ∩, and −.
- Set Operation are
  - Union
  - Intersect
  - Except

- Union

- The UNION operator is used to combine the result-set of two or more SELECT statements.

  - Each SELECT statement within UNION must have the same number of columns
  - The columns must also have similar data types
  - The columns in each SELECT statement must also be in the same order

Syntax:

```
SELECT column_name(s) FROM table1
UNION
SELECT column_name(s) FROM table2;
```

Ex:

```
SELECT FIRST_NAME FROM EMPLOYEE
UNION
SELECT FIRST_NAME FROM CUSTOMER
```

OUTPUT:

Results Explain Describe Saved SQL History

FIRST_NAME
KEERTHI
MANGAL
MITHIL
Palav
Pavan
Santhosh
Veena
Vijaya

8 rows returned in 0.01 seconds Download

- Union All
  - The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL

Syntax

```
SELECT column_name(s) FROM table1
UNION ALL
SELECT column_name(s) FROM table2;
```

Ex:

```
SELECT FIRST_NAME FROM EMPLOYEE
UNION ALL
SELECT FIRST_NAME FROM CUSTOMER
```

OUTPUT:

Results Explain Describe Saved SQL History

FIRST_NAME
MITHIL
MANGAL
KEERTHI
MITHIL
Veena
Vijaya
Pavan
Santhosh
Palav

9 rows returned in 0.00 seconds Download

The Intersect Operation

- The SQL INTERSECT clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement.
- This means INTERSECT returns only common rows returned by the two SELECT statements.
- Just as with the UNION operator, the same rules apply when using the INTERSECT operator.

MySQL does not support the INTERSECT operator.

Syntax:

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]

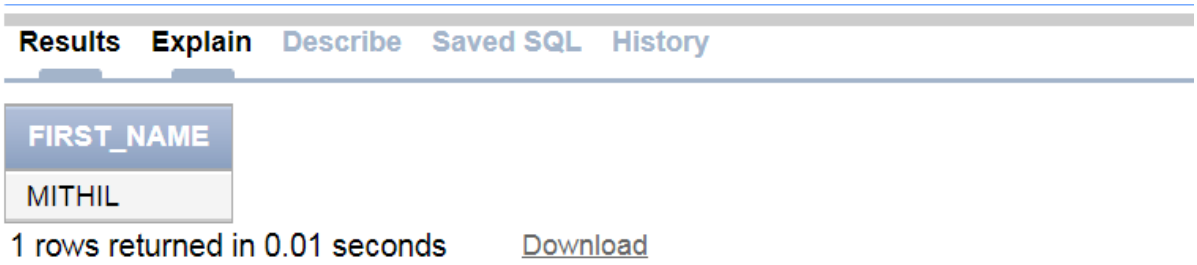
INTERSECT

SELECT column1 [, column2]
FROM table1 [, table2]
[WHERE condition]

Ex:

SELECT FIRST_NAME FROM EMPLOYEE INTERSECT SELECT FIRST_NAME FROM CUSTOMER;

OUTPUT:



The Except Operation

- The SQL EXCEPT clause/operator is used to combine two SELECT statements and returns rows from the first SELECT statement that are not returned by the second SELECT statement.
- This means EXCEPT returns only rows, which are not available in the second SELECT statement.
- Just as with the UNION operator, the same rules apply when using the EXCEPT operator.

Note:

- MySQL does not support the EXCEPT operator.
- In Oracle 11g Except operation is carried out by MINUS Clause.

Syntax

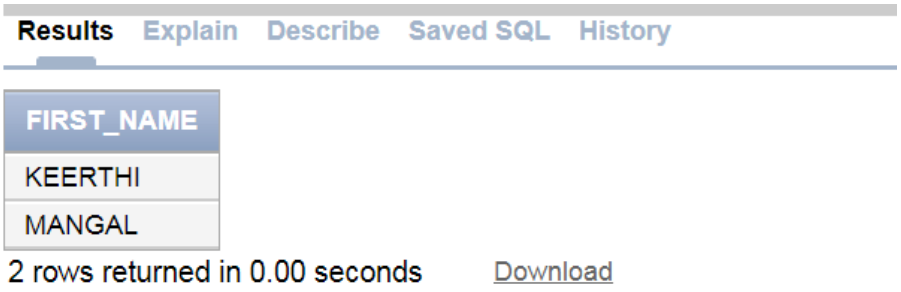
The basic syntax of EXCEPT is as follows.

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

EXCEPT

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```


Ex:

```
SELECT FIRST_NAME FROM EMPLOYEE
MINUS
SELECT FIRST_NAME FROM CUSTOMER
```

OUTPUT:



Null Values

- Null values present special problems in relational operations, including arithmetic operations, comparison operations, and set operations.
- The result of an arithmetic expression (involving, for example +, −, ∗, or /) is null if any of the input values is null.
  Ex:
- If a query has an expression r.A+ 5, and r.A is null for a particular tuple, then the expression result must also be null for that tuple.

Comparisons involving nulls are more of a problem.

Ex:

- consider the comparison “1 < null”.
- It would be wrong to say this is true since we do not know what the null value represents.
- But it would likewise be wrong to claim this expression is false; if we did, “not (1 < null)” would evaluate to true, which does not make sense.
- SQL therefore treats as unknown the result of any comparison involving a null value
- This creates a third logical value in addition to true and false.
- Since the predicate in a where clause can involve Boolean operations such as and, or, and not on the results of comparisons, the definitions of the Boolean operations are extended to deal with the value unknown.

Note :

Since the predicate in a where clause can involve Boolean operations such as and, or, and not on the results of comparisons, the definitions of the Boolean operations are extended to deal with the value unknown.

- and
  The result of true and unknown is unknown, false and unknown is false, while unknown and unknown is unknown.
- or
  The result of true or unknown is true, false or unknown is unknown, while unknown or unknown is unknown.

- not
  The result of not unknown is unknown.

Aggregate Functions

- Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value.
- SQL offers five built-in aggregate functions

  - Average: avg
  - Minimum: min
  - Maximum: max
  - Total: sum
  - Count: count

Note :

The input to sum and avg must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings, as well.

- The COUNT() function returns the number of rows that matches a specified criteria.
- The AVG() function returns the average value of a numeric column.
- The SUM() function returns the total sum of a numeric column.
- The MIN() function returns the smallest value of the selected column.
- The MAX() function returns the largest value of the selected column.

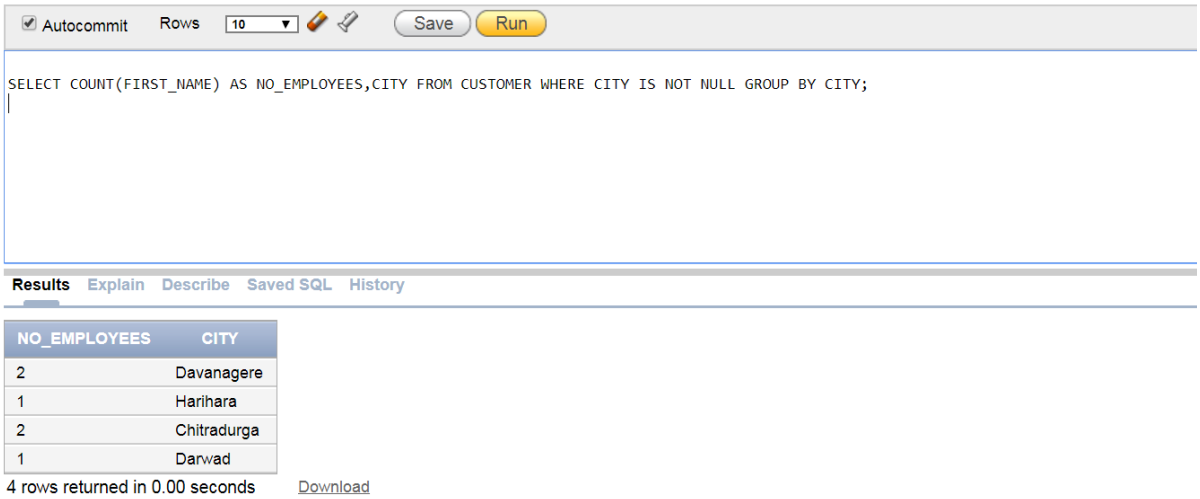
Aggregation with Grouping

- There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples;
- We specify this wish in SQL using the group by clause.
- The attribute or attributes given in the group by clause are used to form groups.
- Tuples with the same value on all attributes in the group by clause are placed in one group.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

Ex:



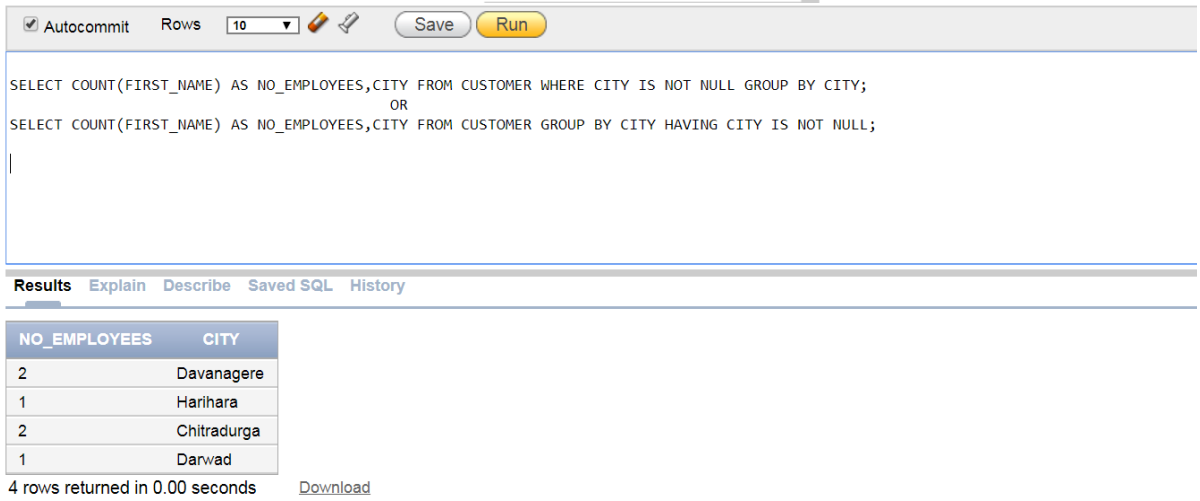
The Having Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

Syntax:

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

Ex:



Aggregation with Null and Boolean Values

- The SQL standard says that the sum operator should ignore null values in its input.
- In general, aggregate functions treat nulls according to the following rule

- All aggregate functions except count (*) ignore null values in their input collection.
- As a result of null values being ignored, the collection of values may be empty.
- The count of an empty collection is defined to be 0, and all other aggregate operations.

Nested Subqueries

- SQL provides a mechanism for nesting subqueries.
- A subquery is a select-from where expression that is nested within another query.
- A common use of subqueries are as below
  - Perform tests for set membership
  - Make set comparisons
  - Determine set cardinality

Note : Subquery can be used in “where” as well as “from” clause.

Set Membership

SQL allows testing tuples for membership in a relation.
The in connective tests for set membership, where the set is a collection of values produced by a select clause.
The not in connective tests for the absence of set membership.

- The SQL IN Operator

  - The IN operator allows you to specify multiple values in a WHERE clause.
  - The IN operator is a shorthand for multiple OR conditions.
  - The IN operator used to retrieve the presence of values.

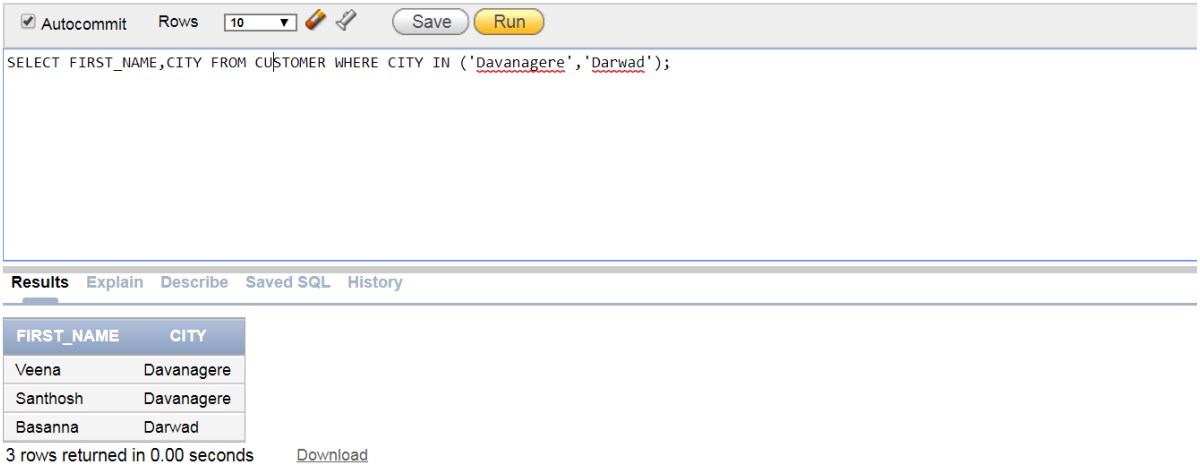
Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

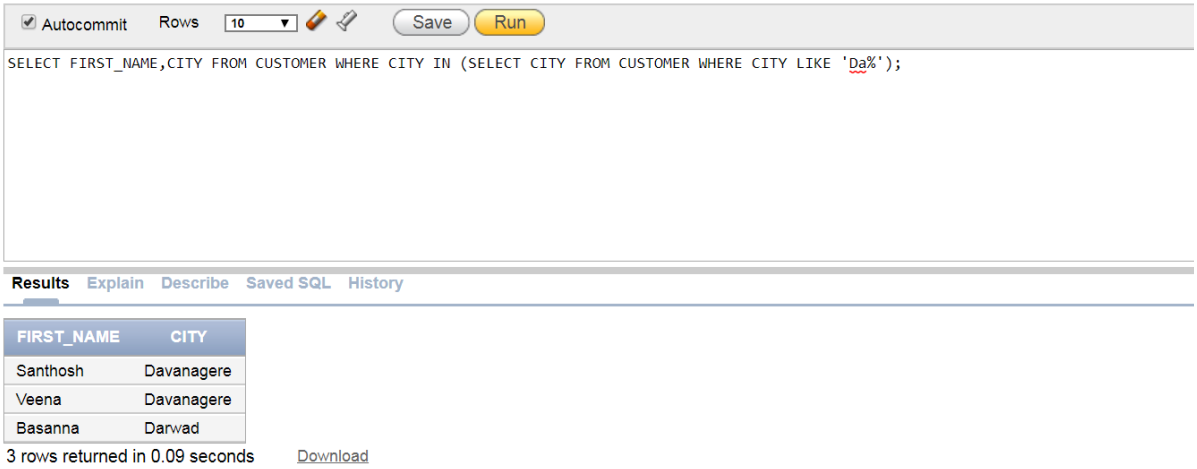

OR

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

Ex: 1



Ex: 2



- The SQL NOT IN Operator

  - The NOT IN operator allows you to specify multiple values in a WHERE clause.
  - The NOT IN operator is a shorthand for multiple OR conditions.
  - The NOT IN operator used to retrieve the absence of values.

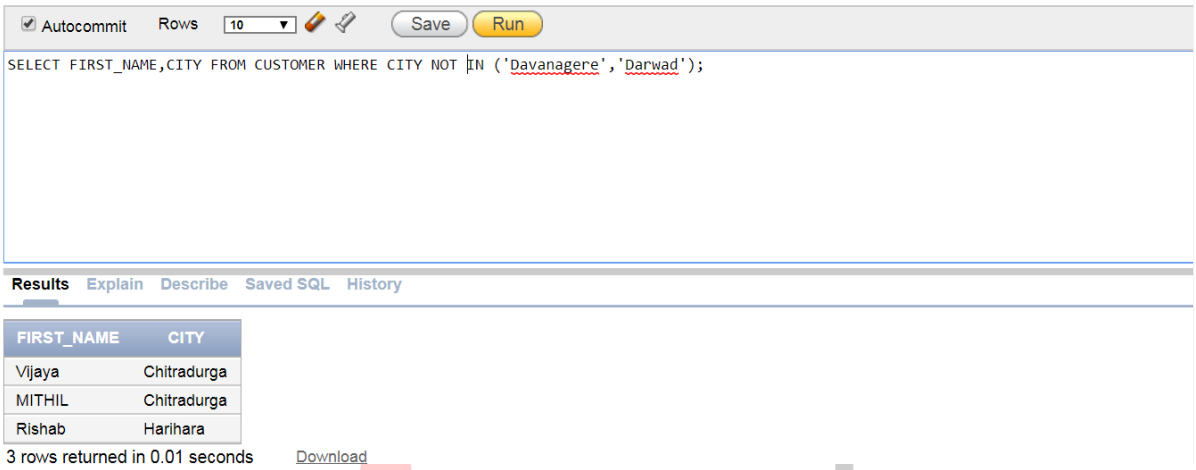
Syntax

SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (value1, value2, ...);

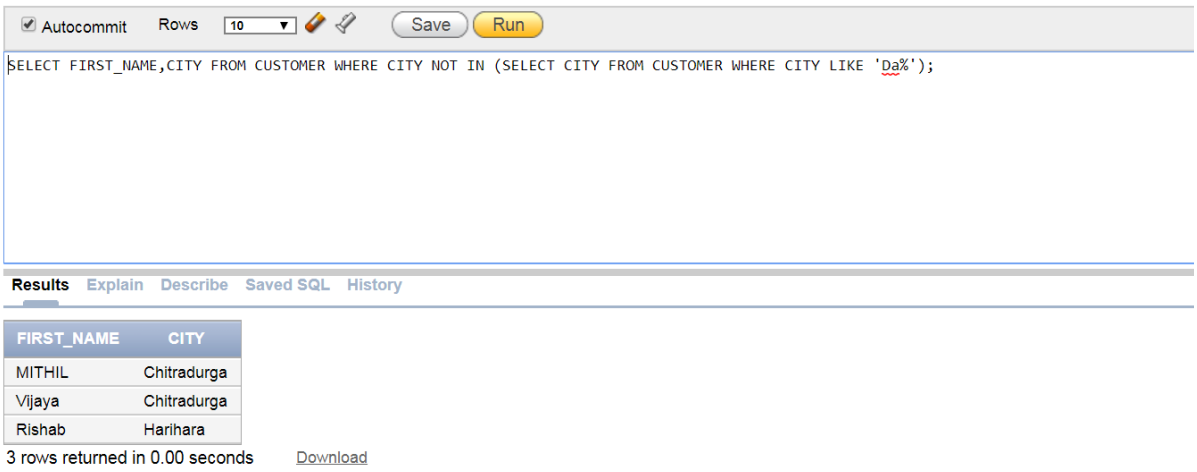
OR

SELECT column_name(s)
FROM table_name
WHERE column_name NOT IN (SELECT STATEMENT);

Ex: 1



Ex: 2



Set Comparison

- As an example of the ability of a nested sub query to compare sets.
- Consider the query “Find the names of all employees whose salary is greater than at least one employee in the Sales department.”

Ex: 1

```
SELECT FIRST_NAME
FROM EMPLOYEE
WHERE SALARY > SOME (SELECT SALARY
FROM EMPLOYEE
```

```
WHERE DEPT_ID =(SELECT DEPT_ID FROM DEPARTMENTS WHERE
DEPT_NAME = 'SALES'));
```

In the above query the sub query is

```
(SELECT SALARY
FROM EMPLOYEE
WHERE DEPT_ID =(SELECT DEPT_ID FROM DEPARTMENTS WHERE
DEPT_NAME = 'SALES'))
```

- Consider the query “Find the names and salary of all employees whose salary is greater than minimum salary.”

```
SELECT FIRST_NAME,SALARY FROM EMPLOYEE WHERE SALARY > (SELECT
MIN(SALARY) FROM EMPLOYEE);
```

In the above example sub query is

```
SELECT MIN(SALARY) FROM EMPLOYEE
```

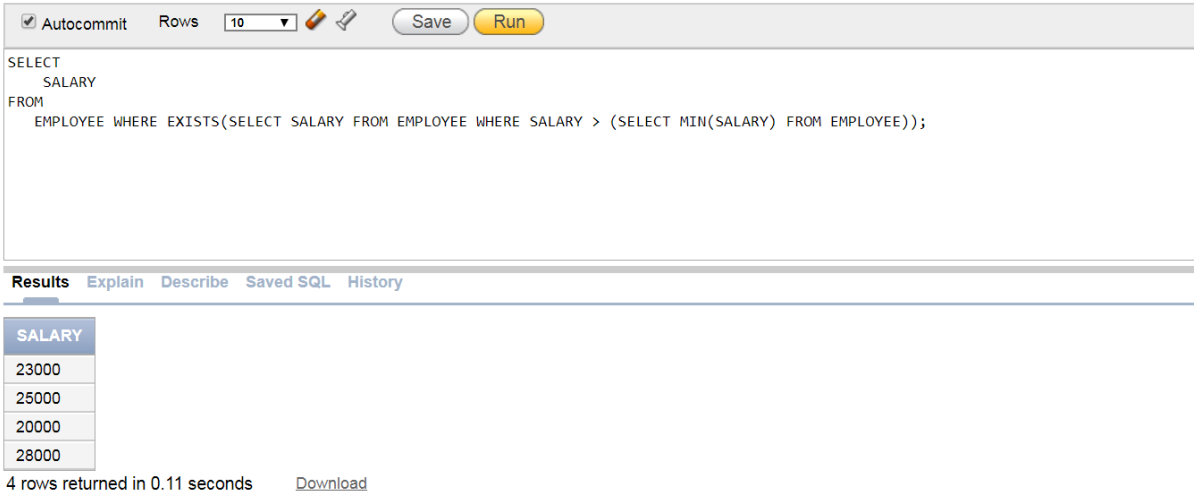
- The > some comparison in the where clause of the outer select is true if the salary value of the tuple is greater than at least one member of the set of all salary values for employee in Sales.
- SQL also allows < some, <= some, >= some, = some, and <> some comparisons.
- As an exercise, verify that = some is identical to in, whereas <> some is not the same as not in.
- As it does for some, SQL also allows < all, <= all, >= all, = all, and <> all comparisons.
- As an exercise, verify that <> all is identical to not in, whereas = all is not the same as in.

Test for Empty Relations

- SQL includes a feature for testing whether a subquery has any tuples in its result.
- The exists construct returns the value true if the argument subquery is nonempty.

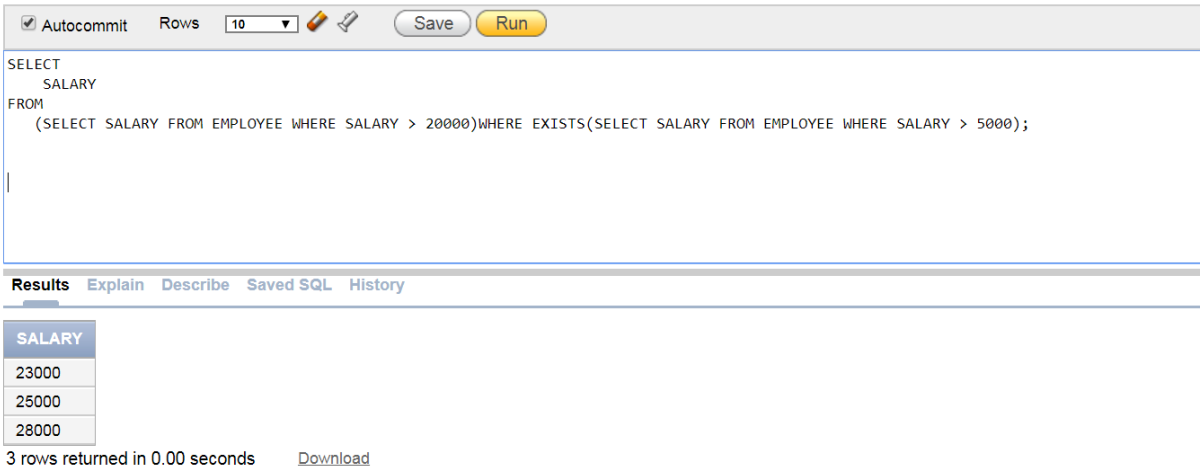
Ex: 1


```
SELECT SALARY FROM EMPLOYEE WHERE EXISTS(SELECT SALARY FROM
EMPLOYEE WHERE SALARY > (SELECT MIN(SALARY) FROM EMPLOYEE));
```



Ex: 2

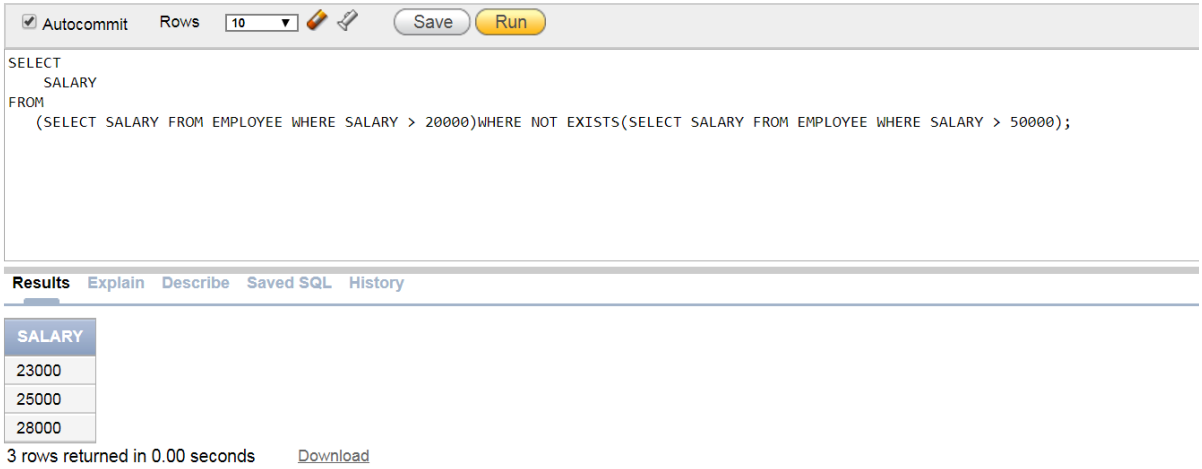
```
SELECT SALARY FROM
(SELECT SALARY FROM EMPLOYEE WHERE SALARY > 20000)WHERE
EXISTS(SELECT SALARY FROM EMPLOYEE WHERE SALARY > 5000);
```



- We can test for the nonexistence of tuples in a subquery by using the not exists construct.

EX 2:

```
SELECT SALARY FROM
(SELECT SALARY FROM EMPLOYEE WHERE SALARY > 20000)WHERE NOT
EXISTS(SELECT SALARY FROM EMPLOYEE WHERE SALARY > 50000);
```



Test for the Absence of Duplicate Tuples

- SQL includes a boolean function for testing whether a subquery has duplicate tuples in its result.
- The unique construct returns the value true if the argument subquery contains no duplicate tuples.

Ex:

Using the unique construct, we can write the query “Find all courses that were offered at most once in 2009” as follows:

```
SELECT T.COURSE ID
FROM COURSE AS T
WHERE UNIQUE (SELECT R.COURSE ID
FROM SECTION AS R
WHERE T.COURSE ID= R.COURSE ID AND
R.YEAR = 2009);
```


- We can test for the existence of duplicate tuples in a subquery by using the not unique construct.

Ex:

To illustrate this construct, consider the query “Find all courses that were offered at least twice in 2009” as follows

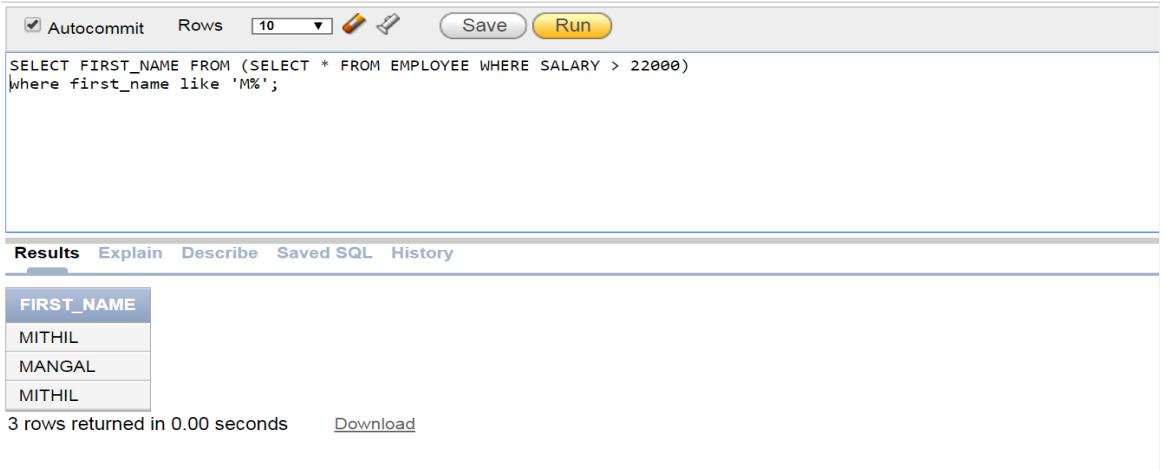
```
SELECT T.COURSE ID
FROM COURSE AS T
WHERE NOT UNIQUE (SELECT R.COURSE ID
FROM SECTION AS R
WHERE T.COURSE ID= R.COURSE ID AND
R.YEAR = 2009);
```

Subqueries in the From Clause

- SQL allows a subquery expression to be used in the from clause.
- The key concept applied here is that any select-from-where expression returns a relation as a result and, therefore, can be inserted into another select-from-where anywhere that a relation can appear.

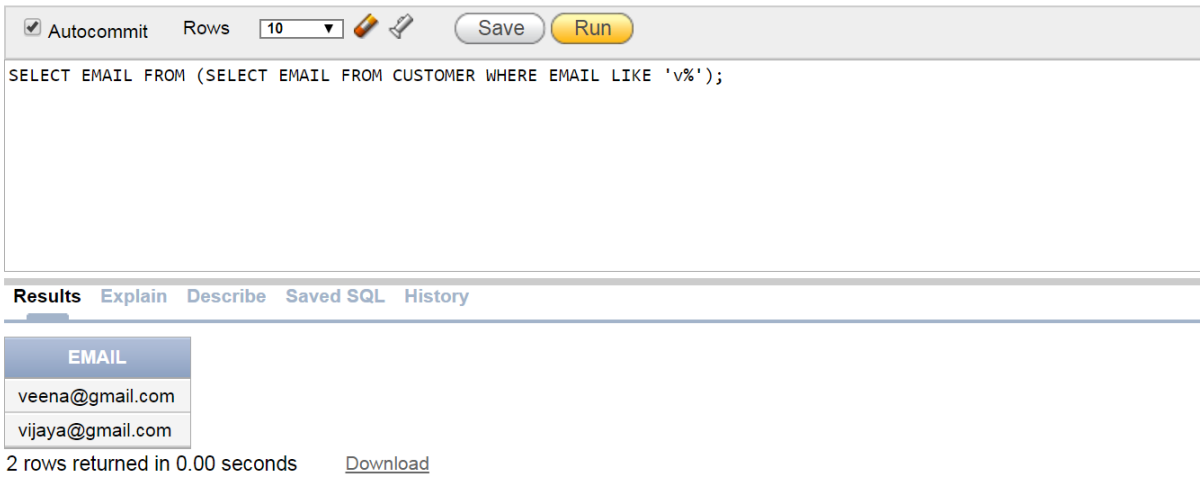
Ex: 1

```
SELECT FIRST_NAME FROM (SELECT * FROM EMPLOYEE WHERE SALARY > 22000) WHERE FIRST_NAME LIKE 'M%';
```



Ex: 2

```
SELECT EMAIL FROM (SELECT EMAIL FROM CUSTOMER WHERE EMAIL LIKE 'v%');
```



The with Clause

- The with clause provides a way of defining a temporary relation whose definition is available only to the query in which the with clause occurs.

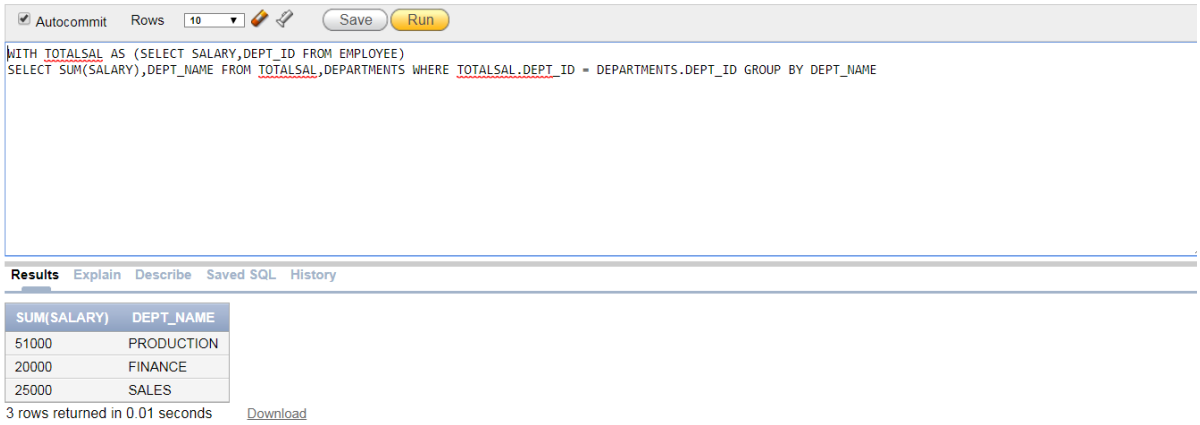
Ex: 1

- Consider the following query, which finds department's wise salary of the employees.

  WITH TOTALSAL AS (SELECT SALARY,DEPT_ID FROM EMPLOYEE)
  SELECT SUM(SALARY),DEPT_NAME FROM TOTALSAL,DEPARTMENTS WHERE TOTALSAL.DEPT_ID = DEPARTMENTS.DEPT_ID GROUP BY DEPT_NAME

- The with clause defines the temporary relation TOTALSAL, which is used in the immediately following query.
- The with clause, introduced in SQL:1999, is supported by many, but not all, database systems.

OUTPUT:



Scalar Subqueries

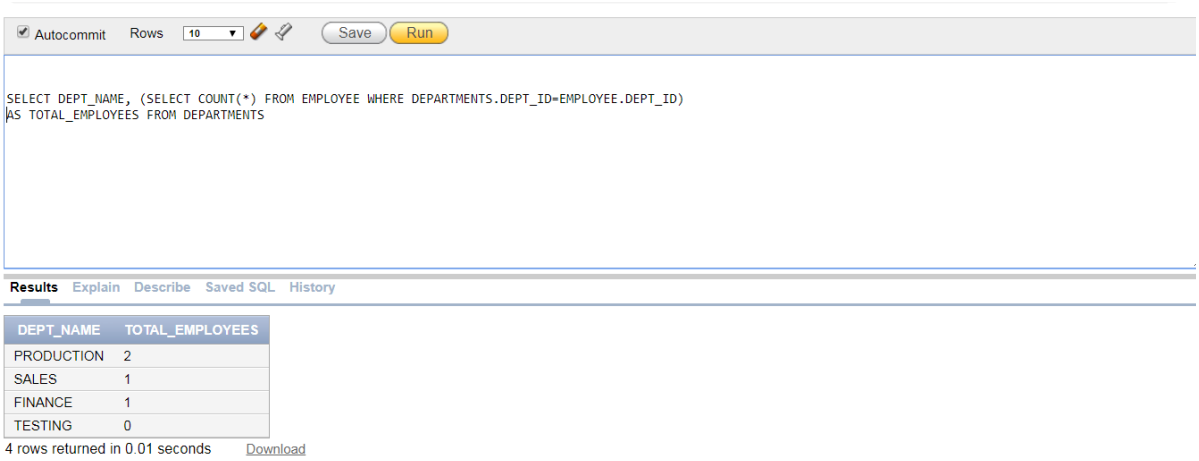
- SQL allows subqueries to occur wherever an expression returning a value is permitted, provided the subquery returns only one tuple containing a single attribute; such subqueries are called scalar subqueries.

- Consider the below query, which returns the total number of employees for each department

  SELECT DEPT_NAME, (SELECT COUNT(*) FROM EMPLOYEE WHERE DEPARTMENTS.DEPT_ID=EMPLOYEE.DEPT_ID) AS TOTAL_EMPLOYEES FROM DEPARTMENTS;

  Here the scalar query is as below.

SELECT COUNT(*) FROM EMPLOYEE WHERE
DEPARTMENTS.DEPT_ID=EMPLOYEE.DEPT_ID



Modification of the Database

- DML is short name of Data Manipulation Language which deals with data manipulation, and includes most common SQL statements such
  - SELECT
  - INSERT
  - UPDATE
  - DELETE
- DML is used to store, modify, retrieve, delete and update data in database.

Deletion

- A delete request is expressed in much the same way as a query.
- We can delete only whole tuples; we cannot delete values on only particular attributes.

- SQL expresses a deletion by

  delete from r
  where P;

Here:

- P represents a predicate and r represents a relation.

- The delete statement first finds all tuples t in r for which P(t) is true, and then deletes them from r.
- The where clause can be omitted, in which case all tuples in r are deleted.

- Note that a delete command operates on only one relation.
- If we want to delete tuples from several relations,we must use one delete command for each relation.
- The predicate in the where clause may be as complex as a select command's where clause. At the other extreme, the where clause may be empty.

- The request delete from employee
- deletes all tuples from the employee relation.
- The employee relation itself still exists, but it is empty.

- Here are examples of SQL delete requests

  Delete all tuples in the employee relation pertaining to employee in the 'J1' JOB_ID.

  ```
  delete from employee
  where JOB_ID= 'J1';
  ```

  Delete all instructors with a salary between 22,000 and 28,000.

  ```
  delete from employee
  where salary between 22000 and 28000;
  ```

Note :

- Although we may delete tuples from only one relation at a time, we may reference any number of relations in a select-from-where nested in the where clause of a delete.
- The delete request can contain a nested select that references the relation from which tuples are to be deleted.

Ex :

Suppose that we want to delete the records of all instructors with salary below the average salary.

```
delete from employee where salary < (select avg (salary) from employee);
```

Insertion

- The INSERT INTO statement is used to insert new records in a table.
- It is possible to write the INSERT INTO statement in two ways.
- The first way specifies both the column names and the values to be inserted

Syntax:

INSERT INTO table_name (column1, column2, column3, ...)
VALUES (value1, value2, value3, ...);

Ex :

INSERT INTO EMPLOYEE(EMP_ID,FIRST_NAME,LAST_NAME,SALARY) VALUES (5,'DEELEP','THIMAYYA',30000);

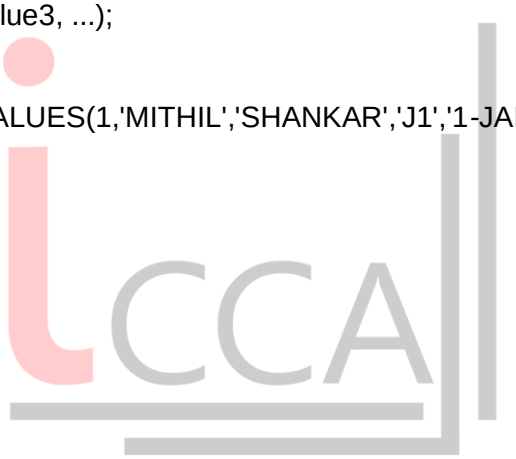
- If we are adding values for all the columns of the table, you do not need to specify the column names in the SQL query.
- However, make sure the order of the values is in the same order as the columns in the table.

The INSERT INTO syntax would be as follows

Syntax:

INSERT INTO table_name
VALUES (value1, value2, value3, ...);

Ex:
INSERT INTO Employee VALUES(1,'MITHIL','SHANKAR','J1','1-JAN-2010',23000,1,1);



Update

The UPDATE statement is used to modify the existing records in a table.

Syntax

```
UPDATE table_name
SET column1 = value1, column2 = value2, ...
WHERE condition;
```

Ex:

UPDATE EMPLOYEE SET DOJ='10/23/2017' WHERE EMP_ID = 5;.

Note: The WHERE clause specifies which record(s) that should be updated. If we omit the WHERE clause, all records in the table will be updated.

SQL Joins

A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

Different Types of SQL JOINs

Here are the different types of the JOINs in SQL

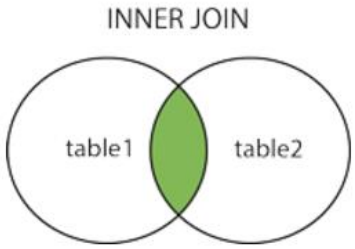
- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Return all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Return all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Return all records when there is a match in either left or right table

Inner join

The INNER JOIN keyword selects records that have matching values in both tables.

Syntax:

```
SELECT column_name(s)
FROM table1
INNER JOIN table2 ON table1.column_name = table2.column_name;
```



Ex :

```
SELECT E.FIRST_NAME,D.DEPT_NAME,E.SALARY FROM EMPLOYEE E INNER
JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID;
```

OUTPUT:

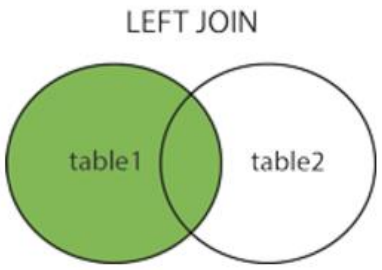
Left join

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

Syntax:

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

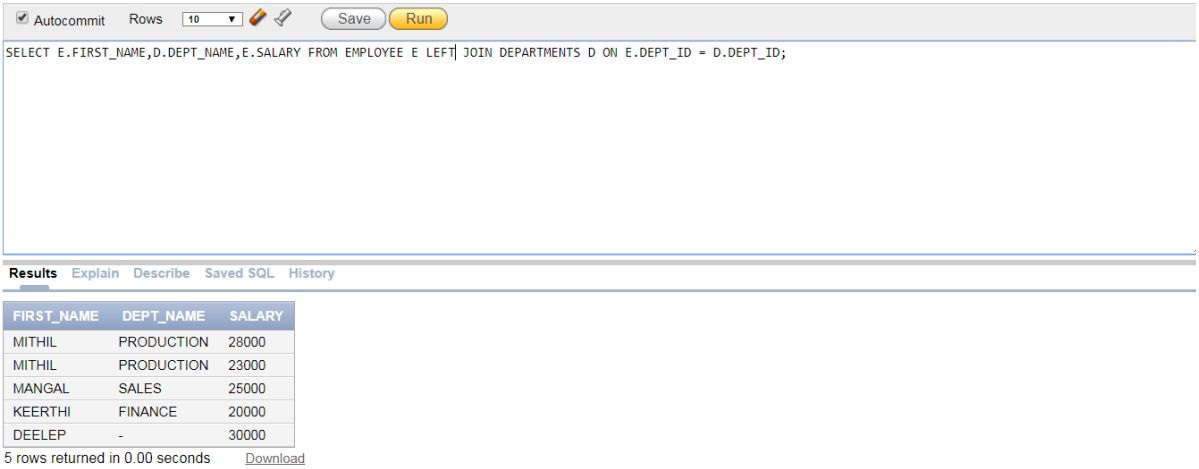
Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.



Ex:

```
SELECT E.FIRST_NAME,D.DEPT_NAME,E.SALARY FROM EMPLOYEE E LEFT
JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID;
```

OUTPUT:



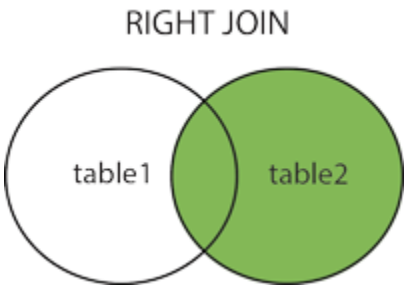
Right join

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

Syntax

SELECT column_name(s)
FROM table1
RIGHT JOIN table2 ON table1.column_name = table2.column_name;

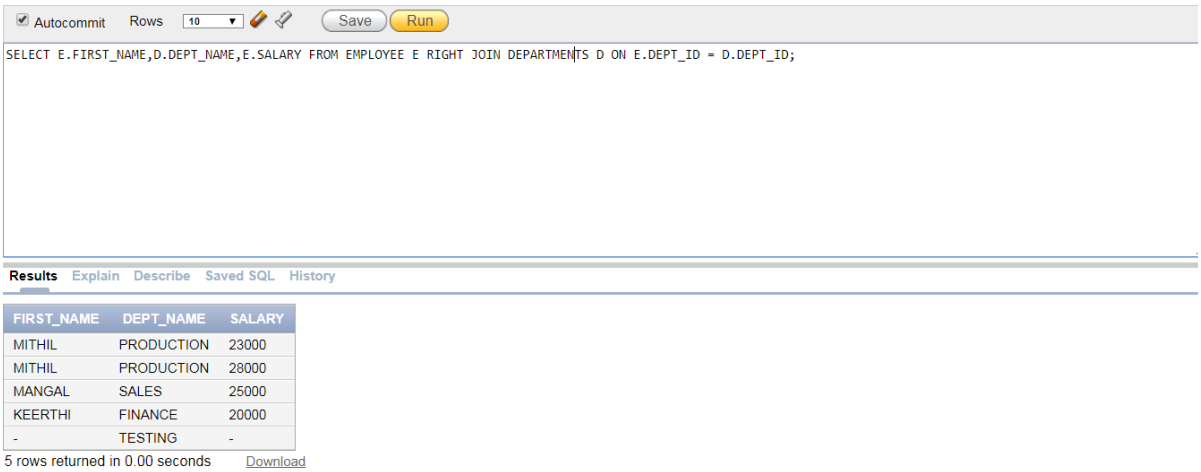
Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



Ex:

SELECT E.FIRST_NAME,D.DEPT_NAME,E.SALARY FROM EMPLOYEE E RIGHT JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID;

OUTPUT:



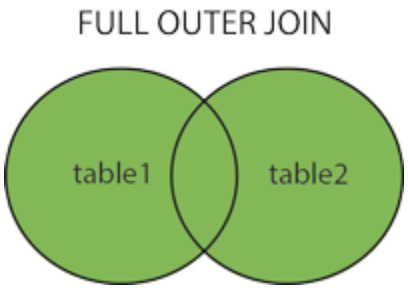
Full join

The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

Syntax:

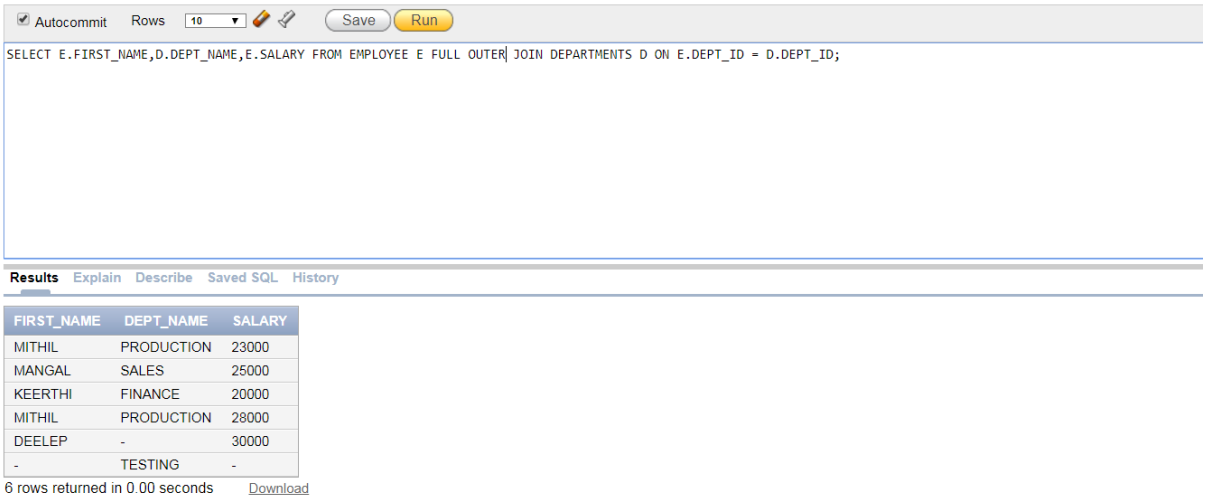
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2 ON table1.column_name = table2.column_name;



Ex:

SELECT E.FIRST_NAME,D.DEPT_NAME,E.SALARY FROM EMPLOYEE E FULL OUTER JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID;

OUTPUT:



Views

- In SQL, a view is a virtual table based on the result-set of an SQL statement.
- A view contains rows and columns, just like a real table.
- The fields in a view are fields from one or more real tables in the database.
- We can add SQL functions, WHERE, and JOIN statements to a view and present the data as if the data were coming from one single table.

Syntax :

```
CREATE VIEW view_name AS
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```


Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

Ex:

```
CREATE VIEW VEMP_DEPT
AS SELECT E.FIRST_NAME,D.DEPT_NAME,E.SALARY FROM EMPLOYEE E INNER
JOIN DEPARTMENTS D ON E.DEPT_ID = D.DEPT_ID;

SELECT * FROM VEMP_DEPT ;
```

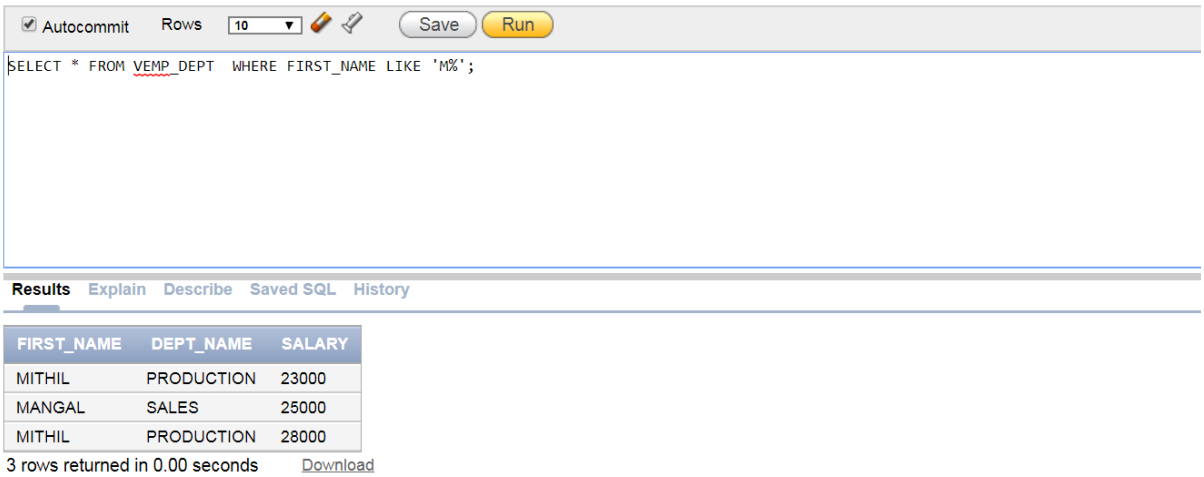
OUTPUT:

Using Views in SQL Queries

Once we have defined a view, we can use the view name to refer to the virtual relation that the view generates.

Using the view VEMP_DEPT below query fetching information whose first_name starts from M.

Ex: SELECT * FROM VEMP_DEPT WHERE FIRST_NAME LIKE 'M%';



Materialized Views

Certain database systems allow view relations to be stored, but they make sure that, if the actual relations used in the view definition change, the view is kept up-to-date.
such views are called materialized views.

Update of a View

In general, an SQL view is said to be updatable (that is, inserts, updates or deletes can be applied on the view) if the following conditions are all satisfied by the query defining the view

- The from clause has only one database relation.
- The select clause contains only attribute names of the relation, and does not have any expressions, aggregates, or distinct specification.
- Any attribute not listed in the select clause can be set to null; that is, it does not have a not null constraint and is not part of a primary key.
- The query does not have a group by or having clause.

Introduction to PL/SQL (brief study)(SQL + PROCEDURES)

- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- It was developed by Oracle Corporation in the early 90's to enhance the capabilities of SQL.
- PL/SQL is one of three key programming languages embedded in the Oracle Database, along with SQL itself and Java.

Following are certain notable points about PL/SQL

- PL/SQL is a completely portable, high-performance transaction-processing language.
- PL/SQL provides a built-in, interpreted and OS independent programming environment.
- PL/SQL can also directly be called from the command-line SQL*Plus interface.
- Direct call can also be made from external programming language calls to database.
- PL/SQL's general syntax is based on that of ADA and Pascal programming language.
- Apart from Oracle, PL/SQL is available in TimesTen in-memory database and IBM DB2.

Structure of PL/SQL

Basic Syntax of PL/SQL which is a block-structured language; this means that the PL/SQL programs are divided and written in logical blocks of code. Each block consists of three sub-parts.

- Declarations
  This section starts with the keyword DECLARE. It is an optional section and defines all variables, cursors, subprograms, and other elements to be used in the program.

- Executable Commands
  This section is enclosed between the keywords BEGIN and END and it is a mandatory section. It consists of the executable PL/SQL statements of the program. It should have at least one executable line of code, which may be just a NULL command to indicate that nothing should be executed.

- Exception Handling

  This section starts with the keyword EXCEPTION. This optional section contains exception(s) that handle errors in the program.

  Every PL/SQL statement ends with a semicolon (;). PL/SQL blocks can be nested within other PL/SQL blocks using BEGIN and END. Following is the basic structure of a PL/SQL block

  Syntax:

```
DECLARE
  <declarations section>
BEGIN
  <executable command(s)>
EXCEPTION
  <exception handling>
END;
```

Unit – III Query Processing Transaction Management

Introduction Transaction Processing

- The concept of transaction provides a mechanism for describing logical units of database processing.
- Transaction processing systems are systems with large databases and hundreds of
- concurrent users executing database transactions.
- Examples of such systems include airline reservations, banking, credit card processing, online retail purchasing, stock markets, supermarket checkouts, and many other applications.
- These systems require high availability and fast response time for hundreds of concurrent users.
- A transaction can be defined as a group of tasks.
- A single task is the minimum processing unit which cannot be divided further.

Ex:

Let's take an example of a simple transaction.
Suppose a bank employee transfers Rs 500 from A's account to B's account.
This very simple and small transaction involves several low-level tasks.

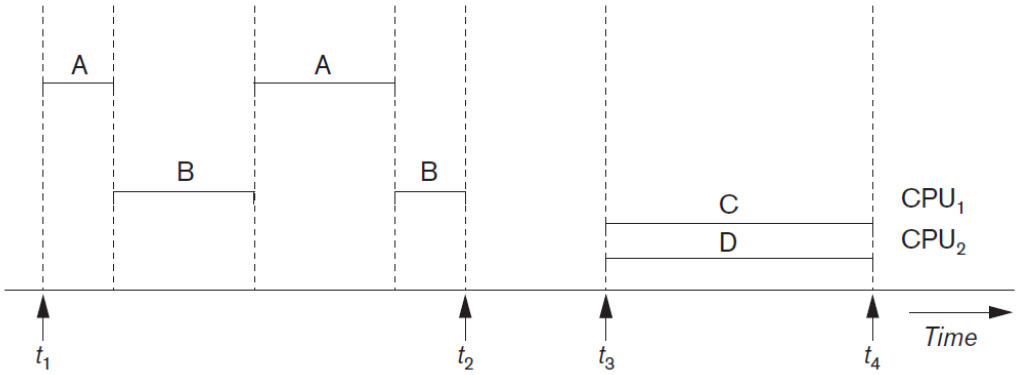
A's Account
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)

B's Account
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)

Single-User versus Multiuser Systems

- Database system can classify according to the number of users use database at an instant.
- A DBMS is single-user if at most one user at a time can use the system, and it is multiuser if many users can use the system— and hence access the database— concurrently.
- Single-user DBMSs are mostly restricted to personal computer systems; most other DBMSs are multiuser.
- For example, an airline reservations system is used by hundreds of users and travel agents concurrently.
- Database systems used in banks, insurance agencies, stock exchanges, supermarkets, and many other applications are multiuser systems.

- In multi user systems, hundreds or thousands of users are typically operating on the database by submitting transactions concurrently to the system.
- Multiple users can access databases—and use computer systems—simultaneously because of the concept of multiprogramming, which allows the operating system of the computer to execute multiple programs—or processes—at the same time.
- A single central processing unit (CPU) can only execute at most one process at a time. However,multiprogramming operating systems execute some commands from one process,then suspend that process and execute some commands from the next process,and so on. A process is resumed at the point where it was suspended whenever it getsits turn to use the CPU again. Hence, concurrent execution of processes is actually interleaved, as illustrated in below figure, which shows two processes, A and B, executing concurrently in an interleaved fashion.
- Interleaving keeps the CPU busy when a process requires an input or output (I/O) operation, such as reading a block from disk.
- The CPU is switched to execute another process rather than remaining idle during I/O time.
- Interleaving also prevents a long process from delaying other processes.
  If the computer system has multiple hardware processors (CPUs), parallel processing of multiple processes is possible, as illustrated by processes C and D in below figure.



Transactions, Database Items, Read and Write Operations, and DBMS Buffers

- A transaction is an executing program that forms a logical unit of database processing.
- A transaction includes one or more database access operations—these can include insertion, deletion, modification (update), or retrieval operations.
- The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.
- One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program; in this case, all database access operations between the two are considered as forming one transaction. A single application program may contain more than one transaction if it contains several transaction boundaries. If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a read-only transaction; otherwise it is known as a read-write transaction.

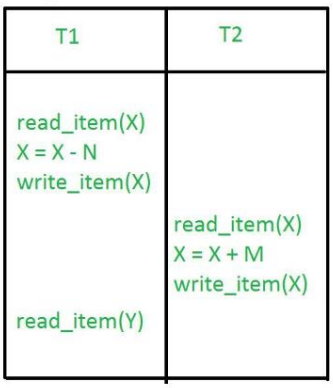
- A database is basically represented as a collection of named data items. The size of a data item is called its granularity.
- A data item can be a database record, but it can also be a larger unit such as a whole disk block, or even a smaller unit such as an individual field (attribute) value of some record in the database.
- The basic database access operations that a transaction can include are as follows
  - read_item(X). Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.
  - write_item(X). Writes the value of program variable X into the database item named X.
  - The basic unit of data transfer from disk to main memory is one disk page (disk block).
  - Executing a read_item(X) command includes the following steps
    1.Find the address of the disk block that contains item X.
    2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer). The size of the buffer is the same as the disk block size.
    3. Copy item X from the buffer to the program variable named X.
  - Executing a write_item(X) command includes the following steps:
    1. Find the address of the disk block that contains item X.
    2. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
    3. Copy item X from the program variable named X into its correct location in the buffer.
    4. Store the updated disk block from the buffer back to disk (either immediately or at some later point in time).

Need of concurrency control

- When multiple transactions execute concurrently in an uncontrolled or unrestricted manner, then it might lead to several problems.
- These problems are commonly referred to as concurrency problems in a database environment. The five concurrency problems that can occur in the database are:
  - Temporary Update Problem
  - Incorrect Summary Problem
  - Lost Update Problem
  - Unrepeatable Read Problem
  - Phantom Read Problem

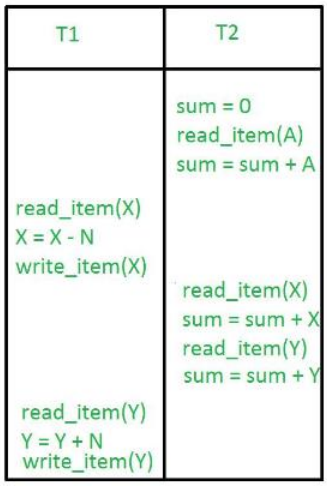
Temporary Update Problem

- Temporary update or dirty read problem occurs when one transaction updates an item and fails.
- But the updated item is used by another transaction before the item is changed or reverted back to its last value.
- In the below example, if transaction 1 fails for some reason then X will revert back to its previous value. But transaction 2 has already read the incorrect value of X.



Incorrect Summary Problem

- Consider a situation, where one transaction is applying the aggregate function on some records while another transaction is updating these records.
- The aggregate function may calculate some values before the values have been updated and others after they are updated.
- In the below example, transaction 2 is calculating the sum of some records while transaction 1 is updating them. Therefore the aggregate function may calculate some values before they have been updated and others after they have been updated.



Lost Update Problem

- In the lost update problem, an update done to a data item by a transaction is lost as it is overwritten by the update done by another transaction.
- In the below example, transaction 2 changes the value of X but it will get overwritten by the write commit by transaction 1 on X (not shown in the image above). Therefore, the update done by transaction 2 will be lost. Basically, the write commit done by the last transaction will overwrite all previous write commits.

T1	T2
read_item(X) X = X + N	X = X + 10 write_item(X)

Unrepeatable Read Problem

- The unrepeatable read problem occurs when two or more read operations of the same transaction read different values of the same variable.
- In the below example, once transaction 2 reads the variable X, a write operation in transaction 1 changes the value of the variable X. Thus, when another read operation is performed by transaction 2, it reads the new value of X which was updated by transaction 1.

T1	T2
Read(X)	
	Read(X)
Write(X)	
	Read(X)

Phantom Read Problem

- The phantom read problem occurs when a transaction reads a variable once but when it tries to read that same variable again, an error occurs saying that the variable does not exist.
- In the below example, once transaction 2 reads the variable X, transaction 1 deletes the variable X without transaction 2's knowledge. Thus, when transaction 2 tries to read X, it is not able to do it.

T1	T2
Read(X)	
	Read(X)
Delete(X)	
	Read(X)

Why Recovery Is Needed

- Whenever a transaction is submitted to a DBMS for execution, the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database, then such transactions are said to be committed else termed aborted.
- In case transactions are aborted then need to rollback the things which made by the failed or incomplete transactions.

Types of Failures

Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution.

A computer failure (system crash)

A hardware, software, or network error occurs in the computer system during transaction execution. Hardward crashes are usually media failures—for example, main memory failure.

A transaction or system error

Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of alogical programming error.3 Additionally, the user may interrupt the transaction during its execution.

Local errors or exception conditions detected by the transaction

During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. An exception condition,4 such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception could be programmed in the transaction itself, and in such a case would not be considered as a transaction failure.

Concurrency control enforcement

The concurrency control method may abort a transaction because it violates serializability or it may abort one or more transactions to resolve a state of deadlock among several transactions. Transactions aborted because of serializability violations or deadlocks are typically restarted automatically at a later time.

Disk failure

Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.

Physical problems and catastrophes

This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

ACID Properties

- A transaction is a very small unit of a program and it may contain several low level tasks.
- A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

- Atomicity
    - This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none.
    - There must be no state in a database where a transaction is left partially completed.
    - States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

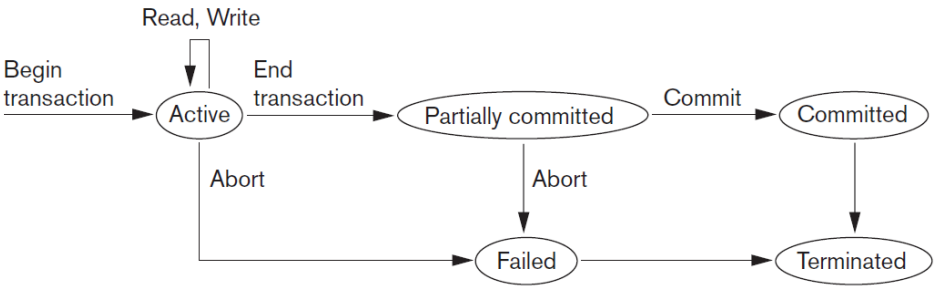
- Consistency
    - The database must remain in a consistent state after any transaction.
    - No transaction should have any adverse effect on the data residing in the database.
    - If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

- Isolation
    - In a database system where more than one transaction are being executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system.
    - No transaction will affect the existence of any other transaction.

- Durability
    - The database should be durable enough to hold all its latest updates even if the system fails or restarts.
    - If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data.
    - If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

States of Transactions

A transaction in a database can be in one of the following states



- Active
  - In this state, the transaction is being executed. This is the initial state of every transaction.
- Partially Committed
  - When a transaction executes its final operation, it is said to be in a partially committed state.
- Failed
  - A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.
- Aborted
  - If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts.
  - Re-start the transaction
  - Kill the transaction
- Committed
  - If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.

Concurrency Control Techniques

Introduction

- In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions.
- We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions.
- Protocol is set of rules and conventions for the communication.
- Concurrency control protocols can be broadly divided into two categories
  - Lock based protocols
  - Time stamp based protocols

Locks

There are various modes in which a data item may be locked

- Shared
  If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then Ti can read, but cannot write, Q.
- Exclusive
  If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then Ti can both read and write Q.

Below matrix shows the behaviour of the shared and exclusive lock

	S	X
S	true	false
X	false	false

Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it.

Locks are of two kinds

- Binary Locks
  - A lock on a data item can be in two states; it is either locked or unlocked.

- Shared/exclusive
  - This type of locking mechanism differentiates the locks based on their uses.
  - If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state.
  - Read locks are shared because no data value is being changed.

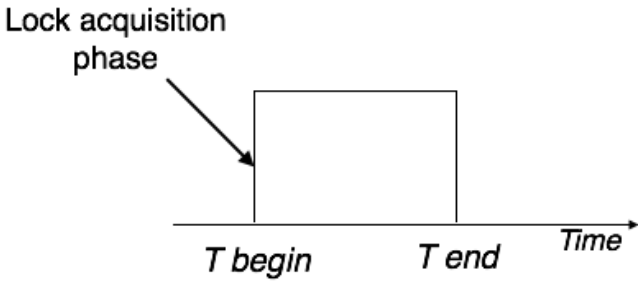
- There are four types of lock protocols available

- Simplistic Lock Protocol

  - Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed.
  - Transactions may unlock the data item after completing the ‘write’ operation.

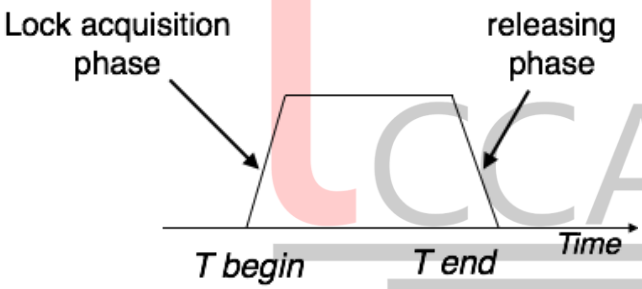
- Pre-claiming Lock Protocol

- Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



- Two-Phase Locking 2PL

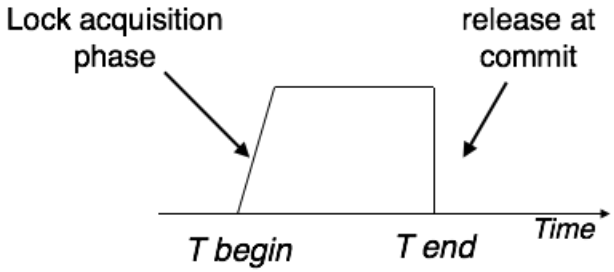
  - This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.



  - Two-phase locking has two phases, one is growing, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.
  - To claim an exclusive (write) lock, a transaction must first acquire a shared (read) lock and then upgrade it to an exclusive lock.

- Strict Two-Phase Locking

  - The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.



Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

- The most commonly used concurrency protocol is the timestamp based protocol.
- This protocol uses either system time or logical counter as a timestamp.
- Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.
- Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction.
- A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.
- In addition, every data item is given the latest read and write-timestamp.
- This lets the system know when the last ‘read and write’ operation was performed on the data item.

Timestamp Ordering Protocol

- The timestamp-ordering protocol ensures serializability among transactions in their conflicting read and write operations.
- This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

Recovery and Atomicity

- When a system crashes, it may have several transactions being executed and various files opened for them to modify the data items.
- Transactions are made of various operations, which are atomic in nature.
- But according to ACID properties of DBMS, atomicity of transactions as a whole must be maintained, that is, either all the operations are executed or none.
- When a DBMS recovers from a crash, it should maintain the following scenarios

    - It should check the states of all the transactions, which were being executed.
    - A transaction may be in the middle of some operation; the DBMS must ensure the atomicity of the transaction in this case.
    - It should check whether the transaction can be completed now or it needs to be rolled back.
    - No transactions would be allowed to leave the DBMS in an inconsistent state.

- There are two types of techniques, which can help a DBMS in recovering as well as maintaining the atomicity of a transaction
    - Maintaining the logs of each transaction, and writing them onto some stable storage before actually modifying the database.
    - Maintaining shadow paging, where the changes are done on a volatile memory, and later, the actual database is updated.

Recovery Algorithm

Recovery algorithm using log records for recovery from transaction failure and a combination of the most recent checkpoint and log records to recover from a system crash.

- Transaction Rollback

  First consider transaction rollback during normal operation (that is, not during recovery from a system crash).
  Rollback of a transaction Ti is performed as follows:

  1.The log is scanned backward, and for each log record of Ti of the form <Ti , Xj , V1, V2> that is found:

  a. The value V1 is written to data item Xj, and
  b. A special redo-only log record<Ti , Xj , V1>is written to the log, where V1 is the value being restored to data item Xj during the rollback. These log records are sometimes called compensation log records. Such records do not need undo information, since we never need to undo such an undo operation.

  2. Once the log record <Ti start> is found the backward scan is stopped, and a log record <Ti abort> is written to the log.

- Recovery After a System Crash

  Recovery actions, when the database system is restarted after a crash, take place in two phases:

  - Redo phase

    In the redo phase, the system replays updates of all transactions by scanning the log forward from the last checkpoint. The log records that are replayed include log records for transactions that were rolled back before system crash, and those that had not committed when the system crash occurred.
    This phase also determines all transactions that were incomplete at the time of the crash, and must therefore be rolled back. Such incomplete transactions would either have been active at the time of the checkpoint, and thus would appear in the transaction list in the checkpoint record, or would have started later; further, such incomplete transactions would have neither a<Ti abort> nor a <Ti commit> record in the log.

  - Undo phase

In the undo phase, the system rolls back all transactions in the undo-list. It performs rollback by scanning the log backward from the end.

- Whenever it finds a log record belonging to a transaction in the undo list, it performs undo actions just as if the log record had been found during the rollback of a failed transaction.
- When the system finds a <Ti start> log record for a transaction Ti in undo-list, it writes a <Ti abort> log record to the log, and removes Ti from undo-list.
- The undo phase terminates once undo-list becomes empty, that is, the system has found <Ti start> log records for all transactions that were initially in undo-list.

Deadlock

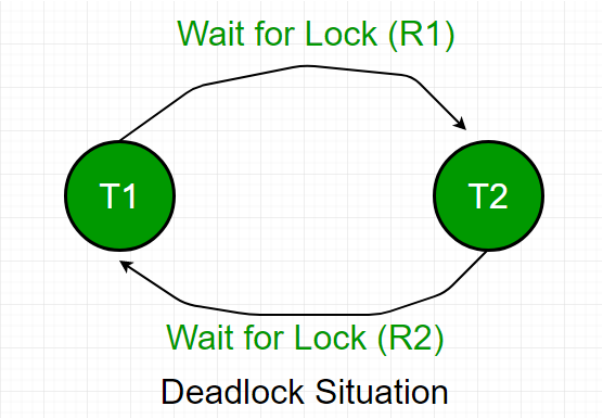
- In a database, a deadlock is an unwanted situation in which two or more transactions are waiting indefinitely for one another to give up locks.
- Deadlock is said to be one of the most feared complications in DBMS as it brings the whole system to a Halt.

Deadlock Avoidance

- When a database is stuck in a deadlock, It is always better to avoid the deadlock rather than restarting or aborting the database. The deadlock avoidance method is suitable for smaller databases whereas the deadlock prevention method is suitable for larger databases.
- One method of avoiding deadlock is using application-consistent logic. In the above-given example, Transactions that access Students and Grades should always access the tables in the same order. In this way, in the scenario described above, Transaction T1 simply waits for transaction T2 to release the lock on Grades before it begins. When transaction T2 releases the lock, Transaction T1 can proceed freely.
- Another method for avoiding deadlock is to apply both row-level locking mechanism and READ COMMITTED isolation level. However, It does not guarantee to remove deadlocks completely.

Deadlock Detection

- When a transaction waits indefinitely to obtain a lock, The database management system should detect whether the transaction is involved in a deadlock or not.
- Wait-for-graph is one of the methods for detecting the deadlock situation. This method is suitable for smaller databases. In this method, a graph is drawn based on the transaction and their lock on the resource. If the graph created has a closed-loop or a cycle, then there is a deadlock.
- For the above-mentioned scenario, the Wait-For graph is drawn below



Deadlock prevention

For a large database, the deadlock prevention method is suitable. A deadlock can be prevented if the resources are allocated in such a way that deadlock never occurs. The DBMS analyzes the operations whether they can create a deadlock situation or not, If they do, that transaction is never allowed to be executed.

Deadlock prevention mechanism proposes two scheme

Wait-Die Scheme

In this scheme, If a transaction requests a resource that is locked by another transaction, then the DBMS simply checks the timestamp of both transactions and allows the older transaction to wait until the resource is available for execution.

Wound Wait Scheme

In this scheme, if an older transaction requests for a resource held by a younger transaction, then an older transaction forces a younger transaction to kill the transaction and release the resource. The younger transaction is restarted with a minute delay but with the same timestamp. If the younger transaction is requesting a resource that is held by an older one, then the younger transaction is asked to wait till the older one releases it.

Starvation

Starvation or Livelock is the situation when a transaction has to wait for an indefinite period of time to acquire a lock.

Reasons for Starvation

- If the waiting scheme for locked items is unfair. (priority queue)
- Victim selection (the same transaction is selected as a victim repeatedly)
- Resource leak.
- Via denial-of-service attack.

Solutions to starvation

- Increasing Priority
  Starvation occurs when a transaction has to wait for an indefinite time, In this situation, we can increase the priority of that particular transaction/s. But the drawback with this solution is that it may happen that the other transaction may have to wait longer until the highest priority transaction comes and proceeds.
- Modification in Victim Selection algorithm
  If a transaction has been a victim of repeated selections, then the algorithm can be modified by lowering its priority over other transactions.
- First Come First Serve approach
  A fair scheduling approach i.e FCFS can be adopted, In which the transaction can acquire a lock on an item in the order, in which the requested the lock.
- Wait-die and wound wait scheme
  These are the schemes that use the timestamp ordering mechanism of transactions.