CS3492 DATABASE MANAGEMENT SYSTEMS L T P C

3 0 0 3

OBJECTIVES:

- To learn the fundamentals of data models, relational algebra and SQL
- To represent a database system using ER diagrams and to learn normalization techniques
- To understand the fundamental concepts of transaction, concurrency and recovery processing
- To understand the internal storage structures using different file and indexing techniques which will help in physical DB design
- To have an introductory knowledge about the Distributed databases, NOSQL and database security

UNIT I RELATIONAL DATABASES 10

Purpose of Database System – Views of data – Data Models – Database System Architecture –

Introduction to relational databases – Relational Model – Keys – Relational Algebra – SQL

fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL

UNIT II DATABASE DESIGN 8

Entity-Relationship model – E-R Diagrams – Enhanced-ER Model – ER-to-Relational Mapping –Functional Dependencies – Non-loss Decomposition – First, Second, Third Normal Forms,

Dependency Preservation – Boyce/Codd Normal Form – Multi-valued Dependencies and Fourth

Normal Form – Join Dependencies and Fifth Normal Form

UNIT III TRANSACTIONS 9

Transaction Concepts – ACID Properties – Schedules – Serializability – Transaction support in SQL– Need for Concurrency – Concurrency control –Two Phase Locking- Timestamp – Multiversion –Validation and Snapshot isolation– Multiple Granularity locking – Deadlock Handling – Recovery Concepts – Recovery based on deferred and immediate update – Shadow paging – ARIES Algorithm

UNIT IV IMPLEMENTATION TECHNIQUES 9

RAID – File Organization – Organization of Records in Files – Data dictionary Storage – Column Oriented Storage– Indexing and Hashing –Ordered Indices – B+ tree Index Files – B tree Index Files – Static Hashing – Dynamic Hashing – Query Processing Overview –

Algorithms for Selection, Sorting and join operations – Query optimization using Heuristics - Cost Estimation.

UNIT V ADVANCED TOPICS 9

Distributed Databases: Architecture, Data Storage, Transaction Processing, Query processing and optimization – NOSQL Databases: Introduction – CAP Theorem – Document Based systems – Key value Stores – Column Based Systems – Graph Databases. Database Security: Security issues – Access control based on privileges – Role Based access control – SQL Injection – Statistical Database security – Flow control – Encryption and Public Key infrastructures – Challenges

TEXT BOOKS:

1. Abraham Silberschatz, Henry F. Korth, S. Sudharshan, "Database System Concepts",

Seventh Edition, McGraw Hill, 2020.

2. Ramez Elmasri, Shamkant B. Navathe, "Fundamentals of Database Systems", Seventh

Edition, Pearson Education, 2017

REFERENCES:

1. C.J.Date, A.Kannan, S.Swamynathan, "An Introduction to Database Systems", Eighth Edition,

Pearson Education, 2006.

PANIMALAR INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

CS3492 DATABASE MANAGEMENT SYSTEMS

UNIT-1 NOTES

RELATIONAL DATABASES

Purpose of Database System – Views of data – Data Models – Database System Architecture – Introduction to relational databases – Relational Model – Keys- Relational Algebra – SQL fundamentals – Advanced SQL features – Embedded SQL– Dynamic SQL

INTRODUCTION

"A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient."

Database-System Applications

Databases are widely used. Here are some applications:

- ***Sales*:** For customer, product, and purchase information.
- ***Accounting*:** For payments, receipts, account balances, assets and other accounting information.
- ***Human resources*:** For information about employees, salaries, payroll taxes, and benefits, and for generation of paychecks.
- ***Manufacturing*:** For management of the supply chain and for tracking production of items in factories, inventories of items in warehouses and stores, and orders for items.
- ***Online retailers*:** For sales data noted above plus online order tracking, generation of recommendation lists, and maintenance of online product evaluations.
- ***Banking and Finance***
    - *Banking*: For customer information, accounts, loans, and banking transactions.
    - *Credit card transactions*: For purchases on credit cards and generation of monthly statements.

- *Finance*: For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds; also for storing real-time market data to enable online trading by customers and automated trading by the firm.

- ***Universities*:** For student information, course registrations, and grades.
- ***Airlines*:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner.
- ***Telecommunication*:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.

PURPOSE OF DATABASE SYSTEMS/ CHARACTERISTICS OF DBMS

✔ The typical file processing system is supported by a conventional operating system.

✔ The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.

✔ A file processing system has a number of major disadvantages.

Data redundancy and inconsistency.

In file processing, every user group maintains its own files for handling its data processing applications.

Example:

Consider the UNIVERSITY database. Here, two groups of users might be the course registration personnel and the accounting office. The accounting office also keeps data on registration and related billing information, whereas the registration office keeps track of student courses and grades. Storing the same data multiple times is called data redundancy. This redundancy leads to several problems.

•Need to perform a single logical update multiple times.

•Storage space is wasted.

•Files that represent the same data may become inconsistent.

Data inconsistency is the various copies of the same data may no larger Agree. Example:

One user group may enter a student's birth date erroneously as JAN-19-1984, whereas the other user groups may enter the correct value of JAN-29-1984. may be reflected in the Music department records but not elsewhere in the system.

Difficulty in accessing data.

Suppose that one of the university clerks needs to find out the names of all students who live within a particular postal-code area. The clerk asks the data-processing department to generate such a list. Because the designers of the original system did not anticipate this request, there is no application program on hand to meet it. There is, however, an application program to generate the list of all students.

The university clerk has now two choices: either obtain the list of all students and extract the needed information manually or ask a programmer to write the necessary application program. Both alternatives are obviously unsatisfactory.

Suppose that such a program is written, and that, several days later, the same clerk needs to trim that list to include only those students who have taken at least 60 credit hours. As expected, a program to generate such a list does not exist. Again, the clerk has the preceding two options, neither of which is satisfactory. The point here is that conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner.

Data isolation. Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

Integrity problems. The data values stored in the database must satisfy certain types of consistency constraints. Suppose the university maintains an account for each department, and records the balance amount in each account. Suppose also that the university requires that the account balance of a department may never fall below zero. Developers enforce these constraints in the system by adding appropriate code in the various application programs.

Atomicity problems. A computer system, like any other device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. Consider a program to transfer $500 from the account balance of department A to the account balance of department B. If a system failure occurs during the execution of the program, it is possible that the $500 was removed from the balance of department A but was not credited to the balance of

department B, resulting in an inconsistent database state.

Concurrent-access anomalies. For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

Example: When several reservation clerks try to assign a seat on an airline flight, the system should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger.

- **Security problems**.

Enforcing security constraints to the file processing system is difficult

VIEWS OF DATA

A database system is a collection of interrelated data and a set of programs that allow users to access and modify these data. A major purpose of a database system is to provide users with an *abstract* view of the data. That is, the system hides certain details of how the data are stored and maintained.

Data Abstraction

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

- **Physical level**. The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.
- **Logical level**. The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. This is referred to as **physical data independence**.
- **View level**. The highest level of abstraction describes only part of the entire database.

Fig: Views of Data

Instances and Schemas

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**.

Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level.

A database may also have several schemas at the view level, sometimes called **subschemas** that describe different views of the database. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

DATABASE MODELS

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system.

Types of data models

a. Hierarchical Model
b. Network Model
c. Entity-relationship Model
d. Relational Model

a. Hierarchical Model

This database model organizes data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the **Root** data, and expands like a tree, adding child nodes to the parent nodes. In this model, a child node will only have a single parent node.

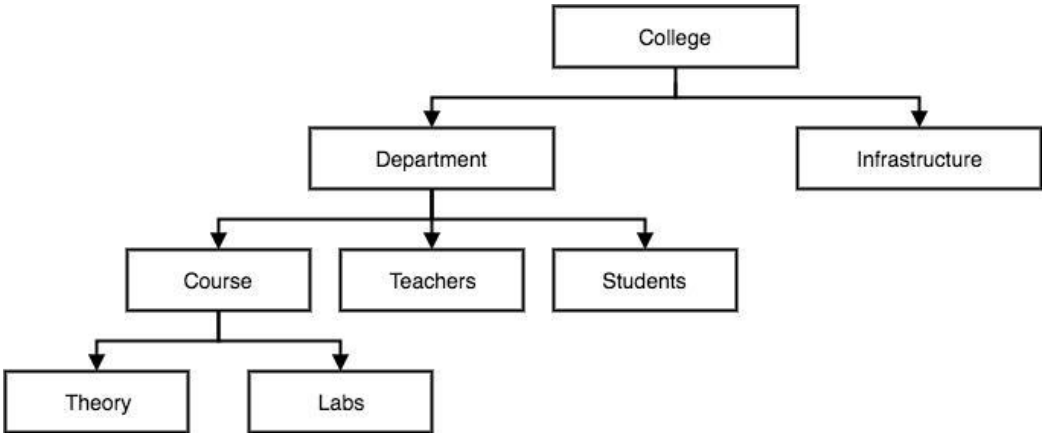


Fig: Hierarchical Model

b. Network Model

This is an extension of the Hierarchical model. In this model data is organized more like a graph, and are allowed to have more than one parent node. This database model was used to map many-to-many data relationships.

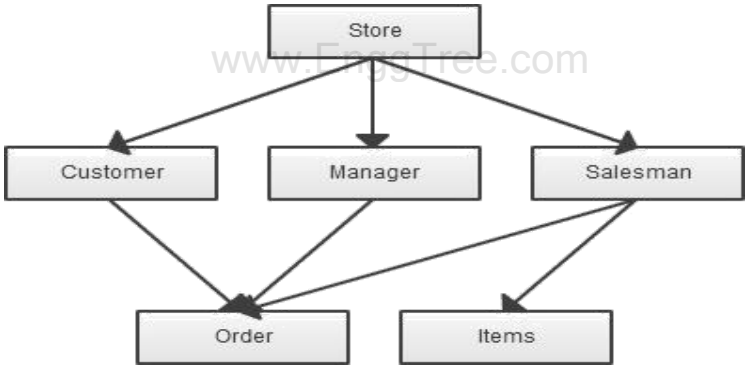


Fig: Network Model

c. Entity-relationship Model

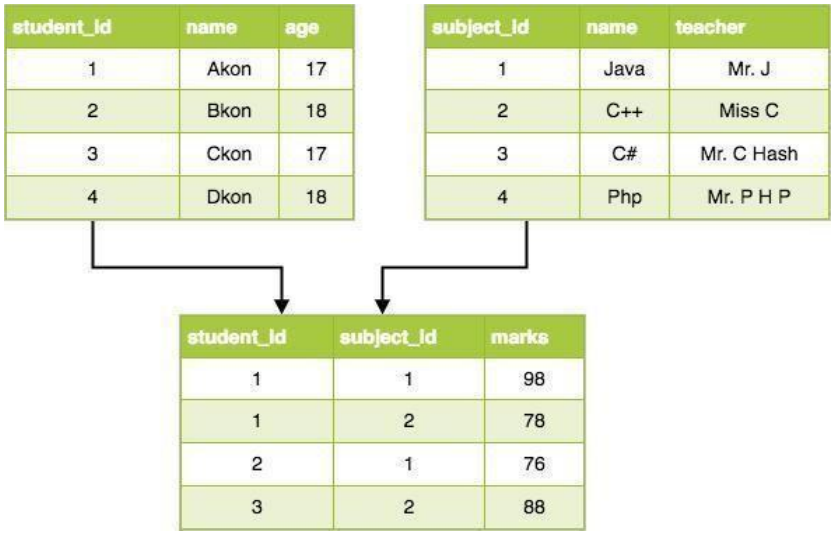
In this database model, relationships are created by dividing object into entity and its characteristics into attributes. Different entities are related using relationships.

Let's take an example, If we have to design a School Database, then **Student** will be an **entity** with **attributes** name, age, address etc. As **Address** is generally complex, it can be another **entity** with **attributes** street name, pincode, city etc, and there will be a relationship between them.

Fig: Entity-relationship Model

d. Relational Model

In this model, data is organized in two-dimensional **tables** and the relationship is maintained by storing a common field. The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table. Hence, tables are also known as **relations** in relational model.



student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

student_id	subject_id	marks
1	1	98
1	2	78
2	1	76
3	2	88

Fig: Relational Model

Object oriented model:

In the object-oriented data model (OODM) both data and their relationships are contained in a single structure known as an object. An object is described by its factual content. An object includes information about relationships between the facts within the object, as well as information about its relationships with other objects. Therefore, the facts within the object are given greater meaning. The OODM is said to be a semantic data model because semantic indicates meaning. The OO data model is based on the following components:

An object is an abstraction of a real-world entity.

Attributes describe the properties of an object.

Concepts of Database Architecture (Tier Architecture)

Database architecture uses programming languages to design a particular type of software for businesses or organizations. Database architecture focuses on the design, development, implementation and maintenance of computer programs that store and organize information for businesses, agencies and institutions. A database architect develops and implements software to meet the needs of users.

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier.

The tiers are classified as follows:

a) 1-tier architecture

b) 2-tier architecture

c) 3-tier architecture

d) 4- n-tier architecture

a) 1-tier architecture

One-tier architecture involves putting all of the required components for a software application or technology on a single server or platform.

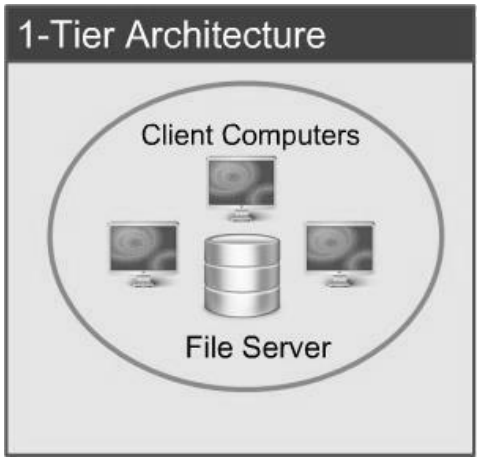


Fig: 1-tier architecture

Basically, a one-tier architecture keeps all of the elements of an application, including the interface, Middleware and back-end data, in one place.

b) 2-tier architecture

The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server.

Advantages

1. Easy to maintain and modification is bit easy.
2. Communication is faster.

Disadvantages

1. In two tier architecture application performance will be degrade upon increasing the users.
2. Cost-ineffective.

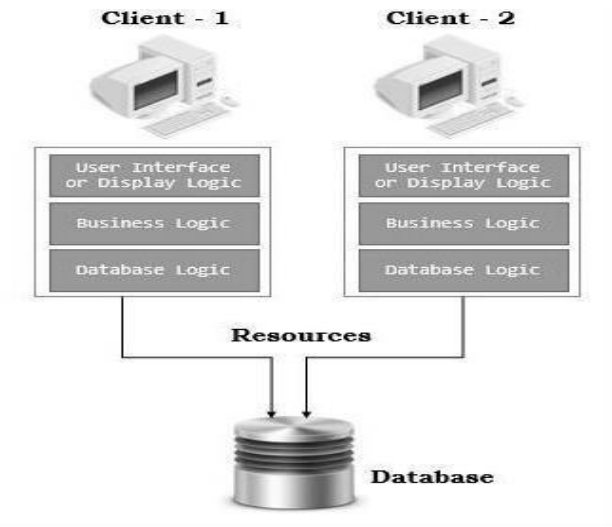


Fig: 2-tier architecture

c) 3-tier architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.

It can be used in web applications and distributed applications.

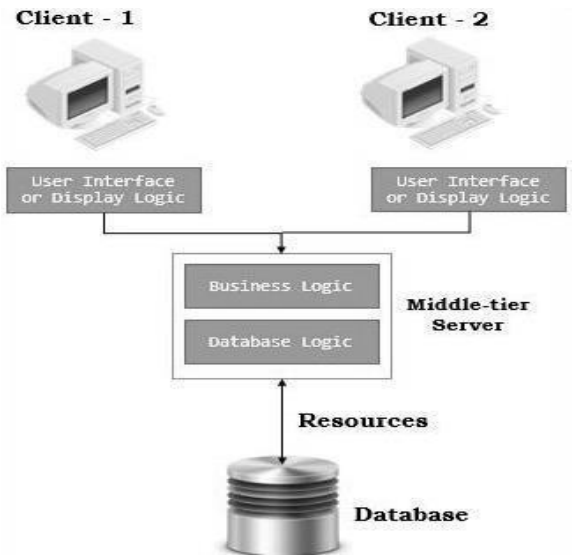


Fig: 3-tier architecture

d) N-tier architecture

N-tier architecture would involve dividing an application into three different tiers. These would be the

1. logic tier,
2. the presentation tier, and
3. The data tier.

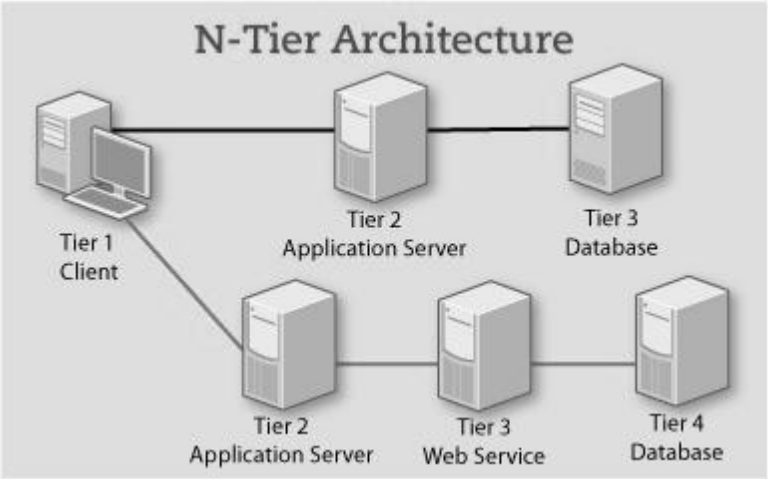


Fig: N-tier architecture

Database System Architecture

DBMS (Database Management System) acts as an interface between the user and the database. The user requests the DBMS to perform various operations such as insert, delete, update and retrieval on the database.

The components of DBMS perform these requested operations on the database and provide necessary data to the users.

Transaction Management

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

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

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

Storage Management

- A storage manager is a program module that provides the interface between the low-level data stored in the database and the application programs and queries submitted to the system.
- The storage manager is responsible for the following tasks:
- Interaction with the file manager

- Efficient storing, retrieving, and Storage Management

Database Administrator

- Coordinates all the activities of the database system; the database administrator has a good understanding of the enterprise's information resources and needs:
- Schema definition
- Storage structure and access method definition
- Schema and physical organization modification
- Granting user authority to access the database
- Specifying integrity constraints
- Monitoring performance and responding to changes in requirements

Database Users

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

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

File manager

manages allocation of disk space and data structures used to represent information on disk.

Database manager

The interface between low level data and application programs and queries.

Query processor

translates statements in a query language into low-level instructions the database manager understands.

The various components of DBMS are described below:

1. **DDL Compiler:**
    - Data Description Language compiler processes schema definitions specified in the DDL.
    - It includes metadata information such as the name of the files, data items, storage details of each file, mapping information and constraints etc.
2. **DML Compiler and Query optimizer:**
    - The DML commands such as insert, update, delete, retrieve from the application program are sent to the DML compiler for compilation into object code for

database access.

- The object code is then optimized in the best way to execute a query by the query optimizer and then send to the data manager.

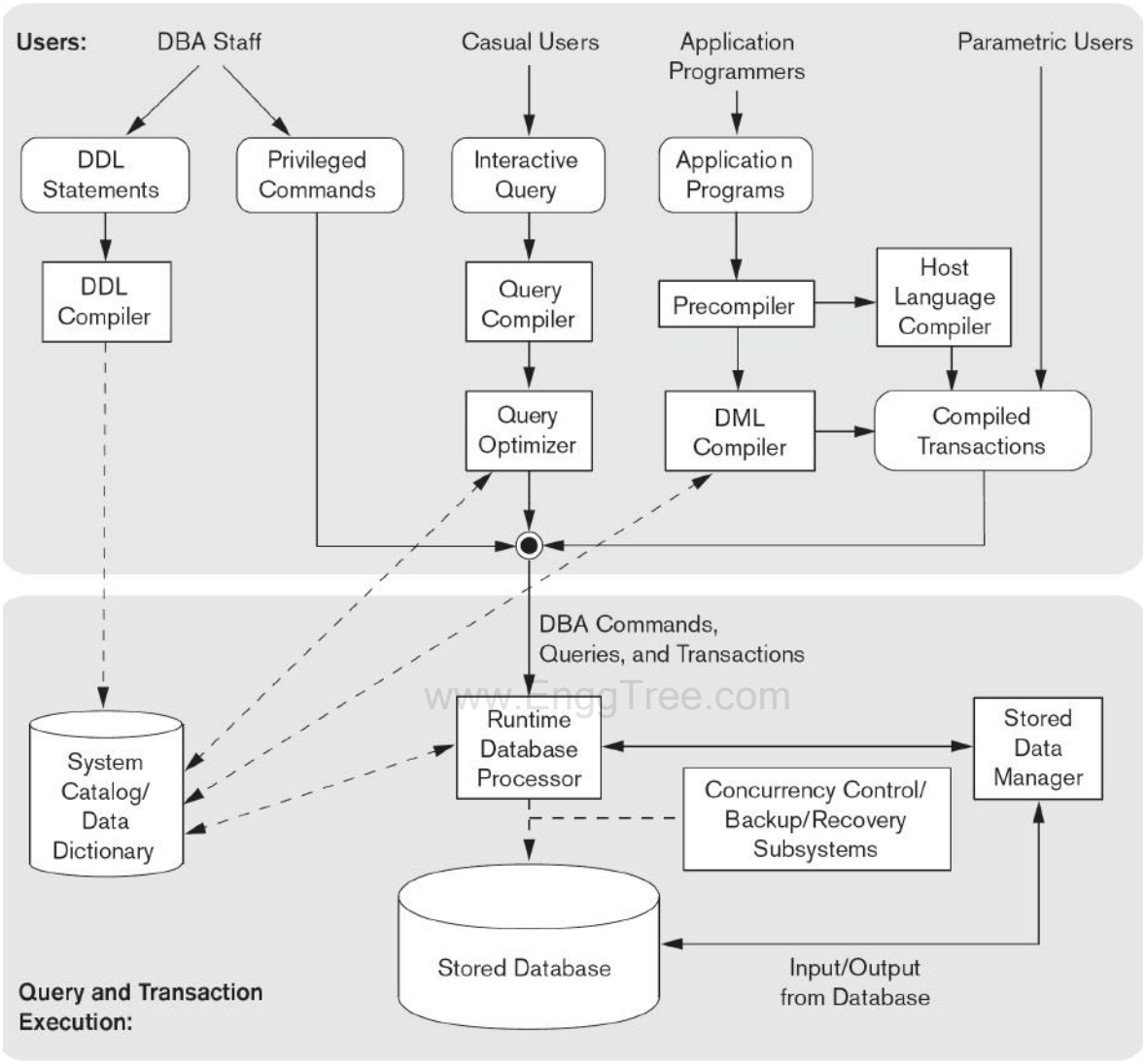


Fig: Component modules of a DBMS and their interactions.

3. Data Manager:

- The Data Manager is the central software component of the DBMS also knows as Database Control System.
- The Main Functions Of Data Manager Are:
  1. Convert operations in user's Queries coming from the application programs or combination of DML Compiler and Query optimizer which is known as Query Processor from user's logical view to physical file system.

2. Controls DBMS information access that is stored on disk.
3. It also enforces constraints to maintain consistency and integrity of the data.
4. It also synchronizes the simultaneous operations performed by the concurrent users.
5. It also controls the backup and recovery operations.

4. **Data Dictionary:**
   - Data Dictionary, stores metadata about the database.
   - Data dictionary is used to actually control the data integrity, database operation and accuracy. It may be used as a important part of the DBMS
5. **Data Files:**
   - Which store the database itself.
6. **Compiled DML:**
   - The DML complier converts the high level Queries into low level file access commands known as compiled DML.
7. **End Users:**
   - The second class of users is end user, who interacts with system from online workstation or terminals.

3. **Query Processor Units:**

a) Machine only understand low level language, so it is the task of query processor to convert user's queries in the series of low level instruction. Then after, it sends these instructions to database manager for execution. There are various component of query processor.

b) DDL Complier: it records the DDL statements into set of tables containing data dictionary. It coverts DDL statement into object form from source form.

c) DML Complier: It converts DML statements into low level instructions that are more easy to understand by query evaluation engine.

d) Query Evaluation Engine: Queries generated by DML compiler are executed in Query evaluation Engine. DDL Interpreter

4. **Storage Manager Units**

a) Checks the authority of users to access data.

b) Checks for the satisfaction of the integrity constraints.

c) Preserves atomicity and controls concurrency.

d) Manages allocation of space on disk.

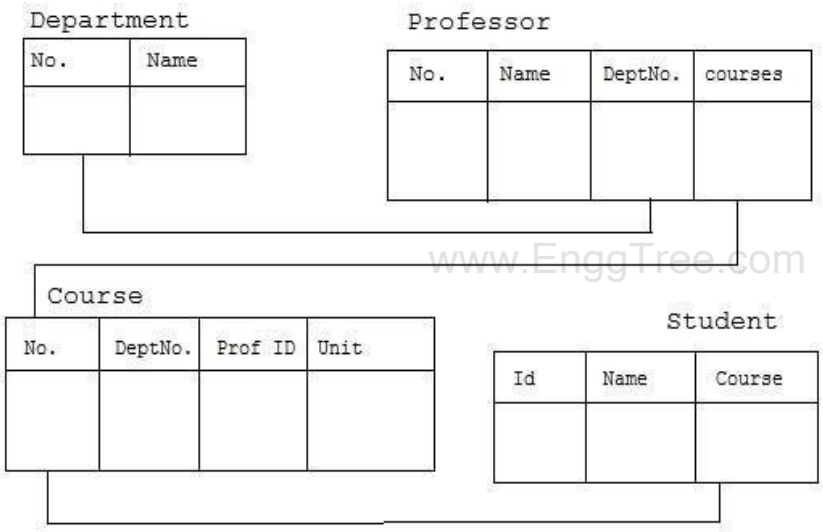
INTRODUCTION TO RELATIONAL DATABASES

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes a DML and DDL.

The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types.

A relational database consists of a collection of tables, each of which is assigned a unique name. A row in a table represents a relationship among a set of values.

RELATIONAL MODEL EXAMPLE



RELATIONAL DATA MODEL IN DBMS: CONCEPTS, CONSTRAINTS, EXAMPLE

The relational model represents the database as a collection of relations. A relation is nothing but a table of values. Every row in the table represents a collection of related data values. These rows in the table denote a real-world entity or relationship.

Some popular Relational Database management systems are:

- DB2 and Informix Dynamic Server - IBM
- Oracle and RDB – Oracle
- SQL Server and Access - Microsoft

Relational Model Concepts

1. **Attribute:** Each column in a Table. Attributes are the properties which define a

relation. e.g., Student_Rollno, NAME,etc.

2. **Tables** – In the Relational model the, relations are saved in the table format. It is stored along with its entities. A table has two properties

rows and columns. Rows represent records and columns represent attributes.

3. **Tuple** – It is nothing but a single row of a table, which contains a single record.
4. **Relation Schema:** A relation schema represents the name of the relation with its attributes.
5. **Degree:** The total number of attributes which in the relation is called the degree of the relation.
6. **Cardinality:** Total number of rows present in the Table.
7. **Column:** The column represents the set of values for a specific attribute.
8. **Relation instance** – Relation instance is a finite set of tuples in the RDBMS system. Relation instances never have duplicate tuples.
9. **Relation key** - Every row has one, two or multiple attributes, which is called relation key.
10. **Attribute domain** – Every attribute has some pre-defined value and scope which is known as attribute domain

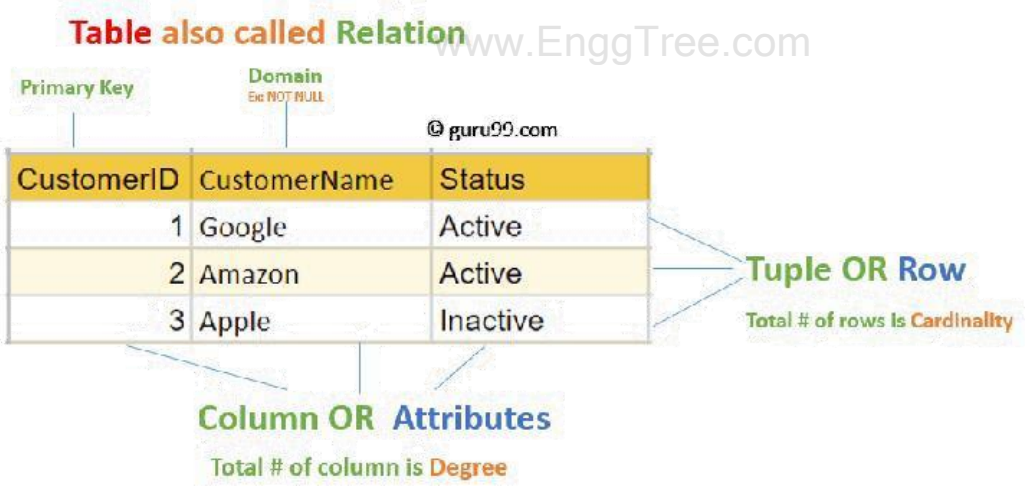


Fig: Relational Model Concepts

Relational Integrity constraints

Relational Integrity constraints is referred to conditions which must be present for a valid relation. These integrity constraints are derived from the rules in the mini-world that the database represents.

Constraints on the Relational database management system are mostly divided into three main categories are:

a) Domain constraints
b) Key constraints
c) Referential integrity constraints

a) Domain Constraints

Domain constraints can be violated if an attribute value is not appearing in the corresponding domain or it is not of the appropriate data type.

Domain constraints specify that within each tuple, and the value of each attribute must be unique. This is specified as data types which include standard data types integers, real numbers, characters, Booleans, variable length strings, etc.

Example:

Create DOMAIN CustomerName CHECK (value not NULL)

The example shown demonstrates creating a domain constraint such that CustomerName is not NULL.

Key constraints

An attribute that can uniquely identify a tuple in a relation is called the key of the table. The value of the attribute for different tuples in the relation has to be unique.

Example:

In the given table, CustomerID is a key attribute of Customer Table. It is most likely to have a single key for one customer, CustomerID =1 is only for the CustomerName =" Google".

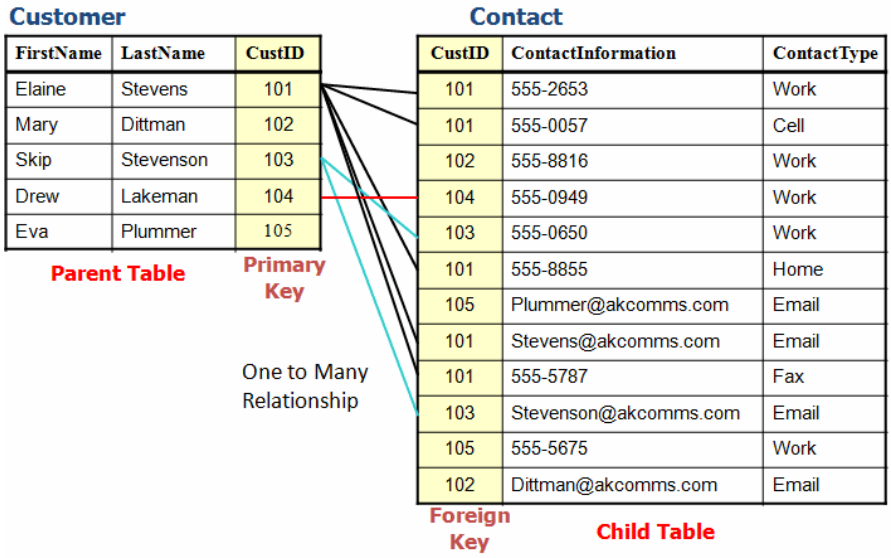


Fig: Relational Integrity constraints

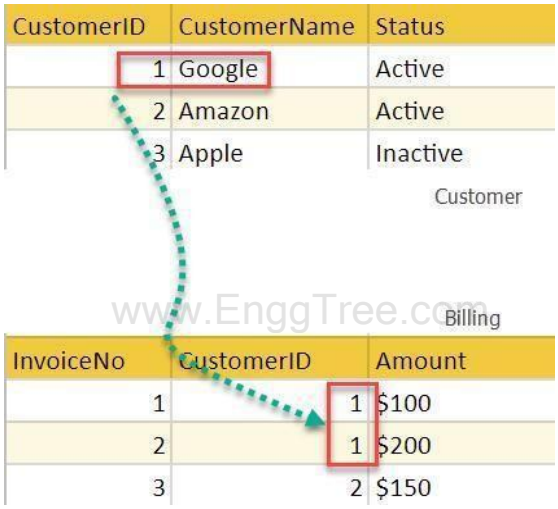
b) Referential integrity constraints

Referential integrity constraints is based on the concept of Foreign Keys. A foreign key is an important attribute of a relation which should be referred to in other relationships. Referential integrity constraint state happens where relation refers to a key attribute of a different or same relation. However, that key element must exist in the table.

Example:

In the below example, we have 2 relations, Customer and Billing.

Tuple for CustomerID =1 is referenced twice in the relation Billing. So we know CustomerName=Google has billing amount $300



CustomerID	CustomerName	Status
1	Google	Active
2	Amazon	Active
3	Apple	Inactive

Customer

Billing

InvoiceNo	CustomerID	Amount
1	1	$100
2	1	$200
3	2	$150

Operations in Relational Model

Insert, update, delete and

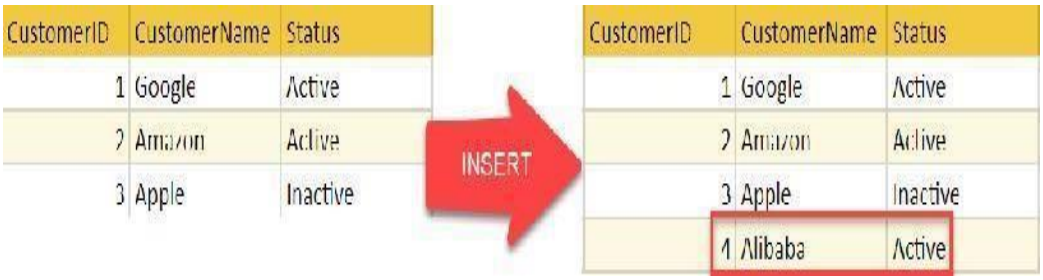
select.

- Insert is used to insert data into the relation
- Delete is used to delete tuples from the table.
- Modify allows you to change the values of some attributes in existing tuples.
- Select allows you to choose a specific range of data.

Whenever one of these operations are applied, integrity constraints specified on the relational database schema must never be violated.

a) Insert Operation

The insert operation gives values of the attribute for a new tuple which should be inserted into a relation.



b) Update Operation

You can see that in the below-given relation table CustomerName= 'Apple' is updated from Inactive to Active.

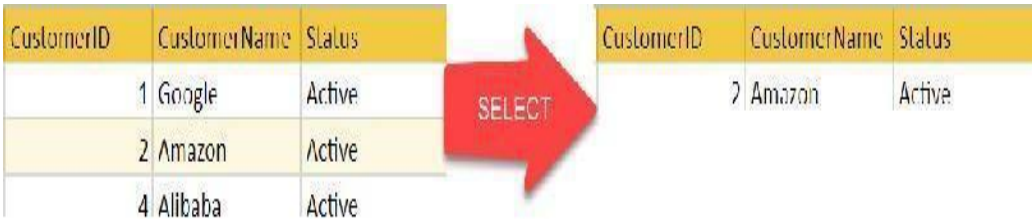
c) Delete Operation

To specify deletion, a condition on the attributes of the relation selects the tuple to be deleted.

In the above-given example, CustomerName= "Apple" is deleted from the table.

The Delete operation could violate referential integrity if the tuple which is deleted is referenced by foreign keys from other tuples in the same database.

d) Select Operation



In the above-given example, CustomerName="Amazon" is selected

Best Practices for creating a Relational Model

- Data need to be represented as a collection of relations
- Each relation should be depicted clearly in the table
- Rows should contain data about instances of an entity
- Columns must contain data about attributes of the entity
- Cells of the table should hold a single value
- Each column should be given a unique name
- No two rows can be identical
- The values of an attribute should be from the same domain

Advantages of using Relational model

- **Simplicity**: A relational data model is simpler than the hierarchical and network model.
- **Structural Independence**: The relational database is only concerned with data and not with a structure. This can improve the performance of the model.
- **Easy to use**: The relational model is easy as tables consisting of rows and columns is quite natural and simple to understand
- **Query capability**: It makes possible for a high-level query language like SQL to avoid complex database navigation.
- **Data independence**: The structure of a database can be changed without having to change any application.
- **Scalable**: Regarding a number of records, or rows, and the number of fields, a database should be enlarged to enhance its usability.

Disadvantages of using Relational model

- Few relational databases have limits on field lengths which can't be exceeded.

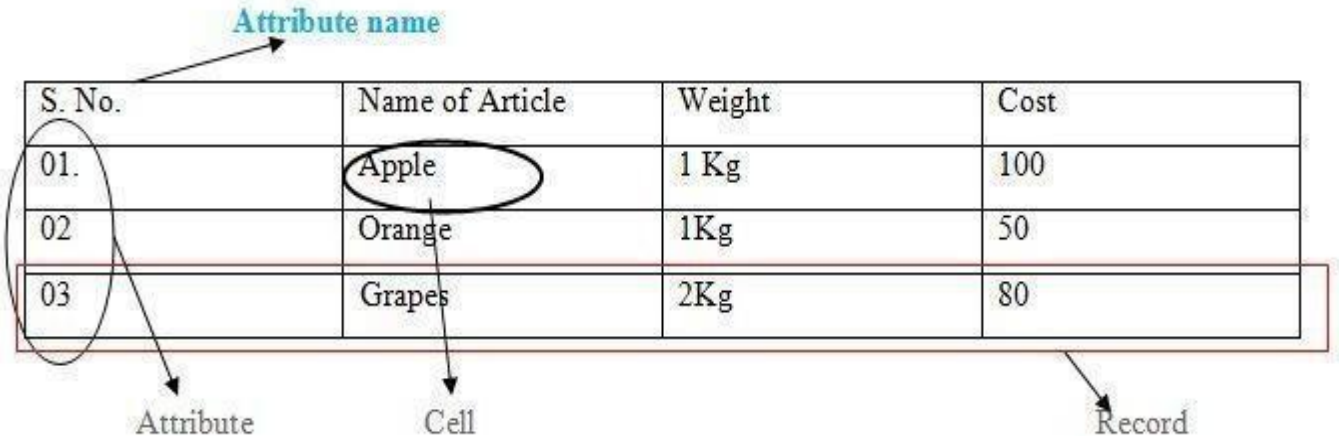
- Relational databases can sometimes become complex as the amount of data grows, and the relations between pieces of data become more complicated.
- Complex relational database systems may lead to isolated databases where the information cannot be shared from one system to another.

Keys in Database Management System (DBMS)

Database table consists of rows and columns, which are technically called **'record or tuple'** and **'attributes or fields'** respectively. A database table is generally called a **'relation'**. The keys are used to uniquely identify a record (row) in the table. Which key should be used depends on requirement.

1. Table = Relation
2. Row = Record/Tuple
3. Column = Attribute/Field

Example:



List of keys in DBMS

1. Super key
2. Candidate key
3. Primary key
4. Composite key

5. Compound key
6. Secondary or Alternative key
7. Non- key attribute
8. Non- prime attribute
9. Foreign key
10. Simple key
11. Artificial key

1) Super keys

Super key is a set of one or more than one columns (attributes) which uniquely identifies each record in a table. Super key is a super set of candidate key.

Roll. No.	First Name of Student	Last Name of Student	Course code
01.	Adam	Gilchrist	A100
02	Alex	Peter	B50
03	John	Lee	C80

For example: Roll No. is unique in relation. This can be selected as a super key. Also we can select more than one column as a super key to uniquely identify a row, like roll no., First name.

2) Candidate keys

Candidate key is a set of one or more than one columns (attributes) which uniquely identifies each record in a table, but there must not be redundant values (repetition of cells) in selected attribute. **Candidate key is a sub set of Super key**.

Roll. No.	First Name of Student	Last Name of Student	Course code
01.	Adam	Gilchrist	A100
02	Adam	Peter	B50
03	John	Gilchrist	C80

For example: Roll No. is unique in relation. This can be selected as a candidate key. Also

we can select more than one column as a candidate key to uniquely identify a record. Unlike the super key in above example we can select only those attributes which don't have repeating cells like course code.

3) Primary keys

Primary key is used to uniquely identify a record in relation. The primary keys are compulsory in every table. The primary keys are having model stability, occurrence of minimum fields, being definitive and feature of accessibility.

Example

Roll. No.	First Name of Student	Last Name of Student	Course code
0111	Adam	Gilchrist	A100
0222	Adam	Peter	B50
0333	John	Gilchrist	C80

Primary Key

Only **Roll No. is unique in the above table, so it is selected as primary key**. Course code can also be selected as a primary key.

4) Composite keys

Composite Key has at-least two or more than two attributes which specially identifies the occurrence of an entity.

Roll. No.	First Name of Student	Last Name of Student	Course code
0111	Adam	Gilchrist	A100
0222	Adam	Peter	B50
0333	John	Gilchrist	C80

Composite key

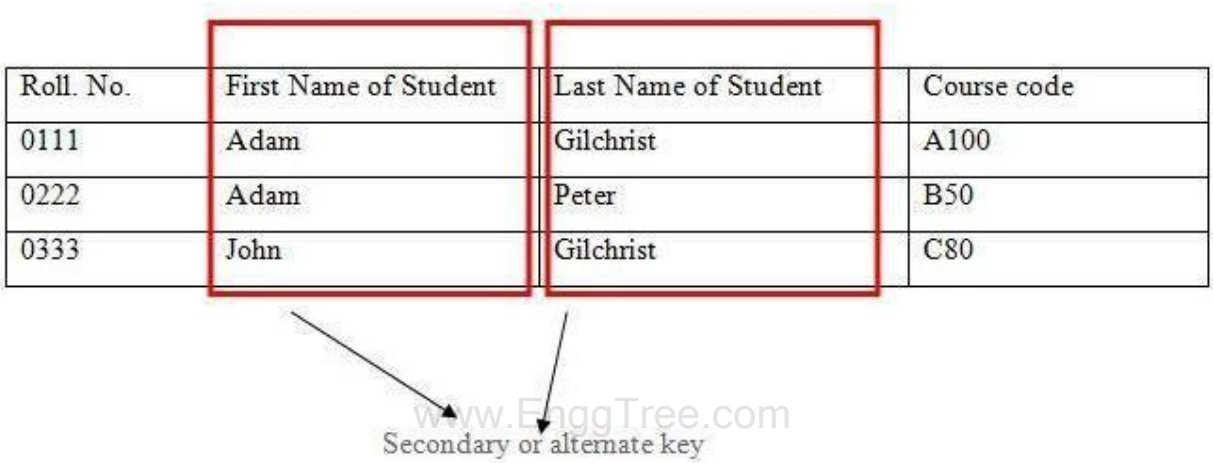
In the above example the **Roll No. and Course Code is combined to uniquely identify the record in relation**.

5) Compound key

Like other keys Compound key is also used to uniquely recognize a record in relation. This can be an attribute or a set of attributes, but the attributes in relation cannot be use as independent keys. If we use them individually, we will not get any unique record.

6) Secondary or Alternative key

The key other than primary keys are called as secondary or alternative keys. Example: If we consider Roll No. and Course code as primary key then First Name of Student and First Name of Student will be Secondary/alternate keys.



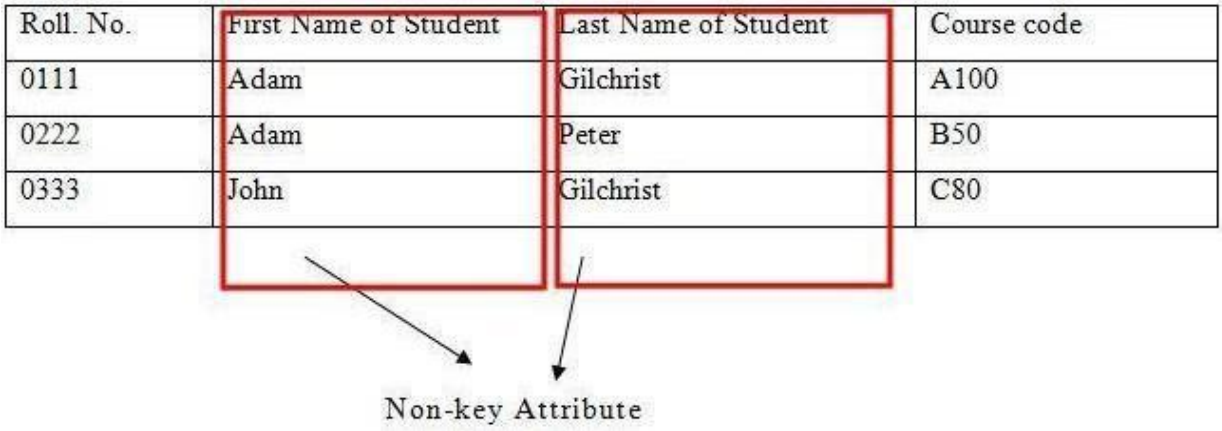
Roll. No.	First Name of Student	Last Name of Student	Course code
0111	Adam	Gilchrist	A100
0222	Adam	Peter	B50
0333	John	Gilchrist	C80

Secondary or alternate key

7) Non-key Attribute

The attributes excluding the candidate keys are called as non-key attributes.

Example: If we consider Roll No. and Course code as candidate key then First Name of Student and First Name of Student will be Non Key attribute.

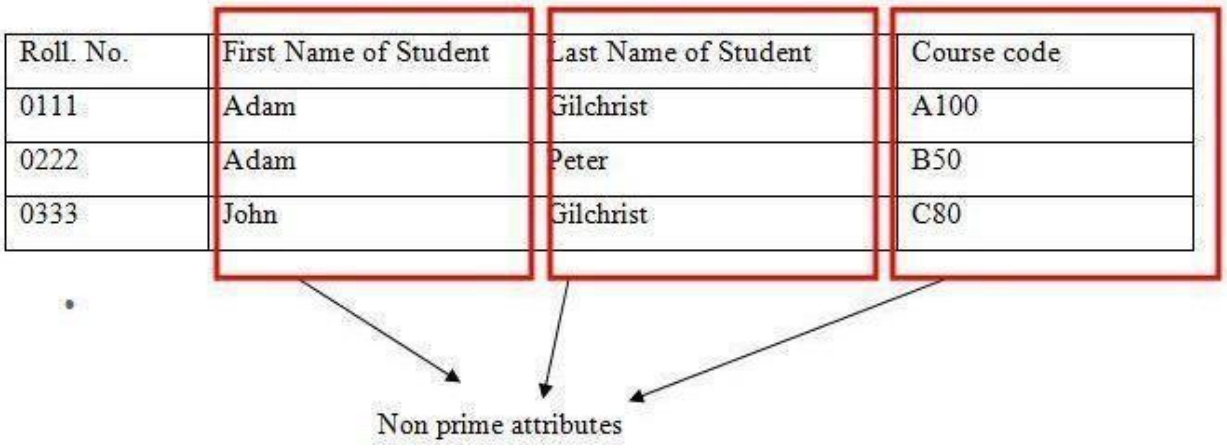


Roll. No.	First Name of Student	Last Name of Student	Course code
0111	Adam	Gilchrist	A100
0222	Adam	Peter	B50
0333	John	Gilchrist	C80

Non-key Attribute

8) Non-prime Attribute

Excluding primary attributes in a table are non-prime attributes.

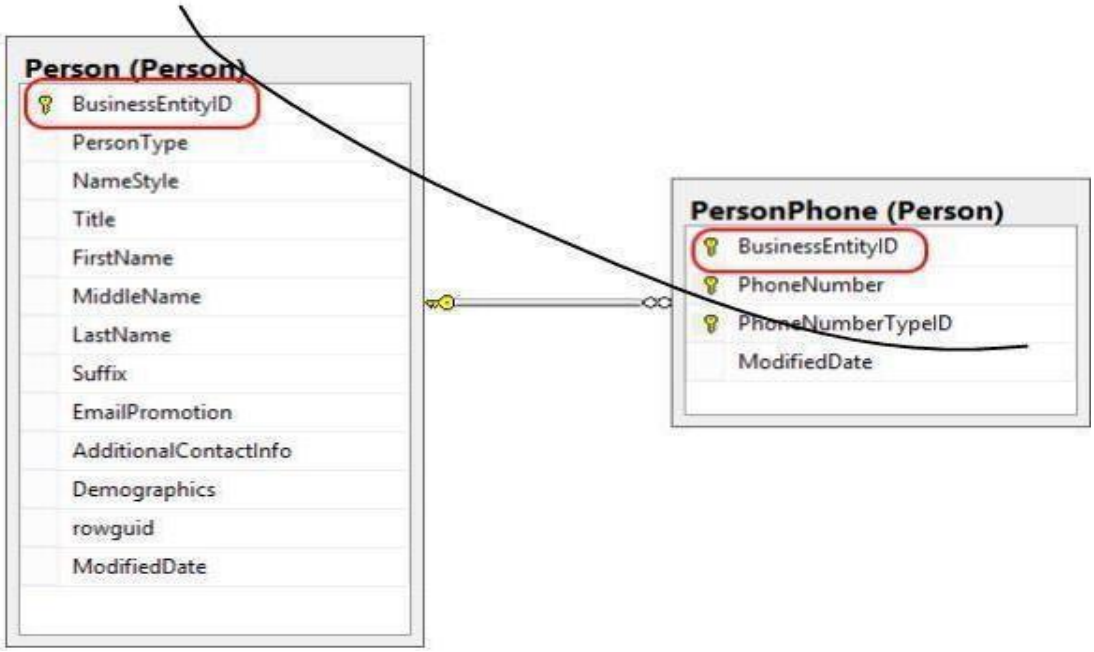


Roll. No.	First Name of Student	Last Name of Student	Course code
0111	Adam	Gilchrist	A100
0222	Adam	Peter	B50
0333	John	Gilchrist	C80

Example: It is considered as only Roll No. is primary key, so all the remaining attributes will be non-prime attributes, but if we considering course code also a primary key than it will not non-prime attribute.

9) Foreign keys

Foreign key is a key of one table, which points to the primary key in second table. It has a relationship with primary key in another table.



The **"BusinessEntityID"** attribute in the **"Person" relation is the PRIMARY KEY**. The **"BusinessEntityID"** attribute in the **"PersonPhone"**relation is a **FOREIGN KEY**.

10) Simple key

Simple key is a single cell to specially identify a record. The single cell cannot be divided into more cells. Primary key is a super set of simple key.

Example: In the below example student id is a single field because no other student will have same Id. Therefore, it is a simple key.

Roll. No.	First Name of Student	Last Name of Student	Course code
0111	Adam	Gilchrist	A100
0222	Adam	Peter	B50
0333	John	Gilchrist	C80

Simple key

11) Artificial key

When primary key is very large and complex, then 'Artificial keys' are used.

RELATIONAL ALGEBRA

Relational algebra is a **procedural** query language that works on relational model. The purpose of a query language is to retrieve data from database or perform various operations such as insert, update, delete on the data.

On the other hand relational calculus is a non-procedural query language, which means it tells what data to be retrieved but doesn't tell how to retrieve it.

Types of operations in relational algebra

1. Basic Operations
2. Derived Operations

Basic/Fundamental Operations:

1. Select (σ)
2. Project (∏)
3. Union (U)
4. Set Difference (-)

5. Cartesian product (X)
6. Rename (ρ)

Derived Operations:

1. Natural Join (⋈)
2. Left, Right, Full outer join (⟕, ⟖, ⟗)
3. Intersection (∩)
4. Division (÷)

1. Select Operator (σ)

Select Operator is denoted by sigma (σ) and it is used to find the tuples (or rows) in a relation (or table) which satisfy the given condition.

Syntax of Select Operator (σ)

σ Condition/Predicate(Relation/Table name)

Select Operator (σ) Example

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

σ Customer_City="Agra" (CUSTOMER)

Output:

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra

C10111	Raghu	Agra

2. Project Operator (∏)

Project operator is denoted by ∏ symbol and it is used to select desired columns (or attributes) from a table (or relation).

Project operator in relational algebra is similar to the Select statement in SQL.

Syntax of Project Operator (∏)

∏ column_name1, column_name2, , column_nameN(table_name)

Project Operator (∏) Example

In this example, we have a table CUSTOMER with three columns, we want to fetch only two columns of the table, which we can do with the help of Project Operator ∏.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

∏ Customer_Name, Customer_City (CUSTOMER)

Output:

Customer_Name	Customer_City
Steve	Agra
Raghu	Agra
Chaitanya	Noida
Ajeet	Delhi
Carl	Delhi

3. Union Operator (∪)

Union operator is denoted by ∪ symbol and it is used to select all the rows (tuples) from two tables (relations).

Let's say we have two relations R1 and R2 both have same columns and we want to select all the tuples(rows) from these relations then we can apply the union operator on these relations.

Note: The rows (tuples) that are present in both the tables will only appear once in the union set. In short you can say that there are no duplicates present after the union operation.

Syntax of Union Operator (∪)

table_name1 ∪ table_name2

Union Operator (∪) Example

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 1: COURSE

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18
S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

Table 2: STUDENT

Query:

∏ Student_Name (COURSE) ∪ ∏ Student_Name (STUDENT)

Output:

Student_Name
Aditya
Carl
Paul
Lucy
Rick
Steve

4. Intersection Operator (∩)

Intersection operator is denoted by ∩ symbol and it is used to select common rows (tuples) from two tables (relations).

Syntax of Intersection Operator (∩)

table_name1 ∩ table_name2

Intersection Operator (∩) Example

Course_Id	Student_Name	Student_Id
C101	Aditya	S901
C104	Aditya	S901
C106	Steve	S911
C109	Paul	S921
C115	Lucy	S931

Table 1: COURSE

Student_Id	Student_Name	Student_Age
S901	Aditya	19
S911	Steve	18

S921	Paul	19
S931	Lucy	17
S941	Carl	16
S951	Rick	18

Table 2: STUDENT

Query:

∏ Student_Name (COURSE) ∩ ∏ Student_Name (STUDENT)

Output:

Student_Name
Aditya
Steve
Paul
Lucy

5. Set Difference (-)

Set Difference is denoted by – symbol. Let's say we have two relations R1 and R2 and we want to select all those tuples(rows) that are present in Relation R1 but **not** present in Relation R2, this can be done using Set difference R1 – R2.

Syntax of Set Difference (-)

table_name1 - table_name2

Query:

Let's write a query to select those student names that are present in STUDENT table but not present in COURSE table.

∏ Student_Name (STUDENT) - ∏ Student_Name (COURSE)

Output:

Student_Name
Carl

Rick

6. Cartesian product (X)

Cartesian product is denoted by X symbol. Let's say we have two relations R1 and R2 then the Cartesian product of these two relations (R1 X R2) would combine each tuple of first relation R1 with the each tuple of second relation R2. I know it sounds confusing but once we take an example of this, you will be able to understand this.

Syntax of Cartesian product (X)

R1 X R2

Cartesian product (X) Example

Col_A	Col_B
AA	100
BB	200
CC	300

Table 1: R

Col_X	Col_Y
XX	99
YY	11
ZZ	101

Table 2: S

Query:

Let's find the Cartesian product of table R and S.

R X S

Output:

Col_A	Col_B	Col_X	Col_Y
AA	100	XX	99
AA	100	YY	11
AA	100	ZZ	101
BB	200	XX	99
BB	200	YY	11
BB	200	ZZ	101
CC	300	XX	99
CC	300	YY	11
CC	300	ZZ	101

Note: The number of rows in the output will always be the cross product of number of rows in each table. In our example table 1 has 3 rows and table 2 has 3 rows so the output has 3×3 = 9 rows.

7. Rename (ρ)

Rename (ρ) operation can be used to rename a relation or an attribute of a relation.

Syntax:

ρ(new_relation_name, old_relation_name)

Rename (ρ) Example

Let's say we have a table customer, we are fetching customer names and we are renaming the resulted relation to CUST_NAMES.

Table: CUSTOMER

Customer_Id	Customer_Name	Customer_City
C10100	Steve	Agra
C10111	Raghu	Agra
C10115	Chaitanya	Noida
C10117	Ajeet	Delhi
C10118	Carl	Delhi

Query:

ρ(CUST_NAMES, ∏(Customer_Name)(CUSTOMER))

Output:

CUST_NAMES
Steve
Raghu
Chaitanya
Ajeet
Carl

8. Joins

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.

Types of join

- ✔ **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

$R1 \bowtie_\theta R2$

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 $R1 \cap R2 = \Phi$.

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 $\bowtie_{\text{Student.Std = Subject.Class}}$ 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.

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

SQL FUNDAMENTALS

SQL | DDL, DQL, DML, DCL and TCL Commands

Structured Query Language(SQL) is the database language which can perform certain operations on the existing database and also we can use this language to create a database. SQL uses certain commands like Create, Drop, Insert etc. to carry out the required tasks.

These SQL commands are mainly categorized into four categories as:

1. DDL – Data Definition Language
2. DQl – Data Query Language
3. DML – Data Manipulation Language
4. DCL – Data Control Language

1. **DDL(Data Definition Language) :** DDL or Data Definition Language actually consists of the SQL commands that can be used to define the database schema. It simply deals with descriptions of the database schema and is used to create and modify the structure of database objects in the database.

Examples of DDL commands:

- **CREATE** – is used to create the database or its objects (like table, index, function, views, store procedure and triggers).
- **DROP** – is used to delete objects from the database.
- **ALTER-**is used to alter the structure of the database.
- **TRUNCATE**–is used to remove all records from a table, including all spaces allocated for the records are removed.
- **COMMENT** –is used to add comments to the data dictionary.
- **RENAME** –is used to rename an object existing in the database.

2. **DQL (Data Query Language) :**

DML statements are used for performing queries on the data within schema objects. The purpose of DQL Command is to get some schema relation based on the query passed to it.

Example of DQL:

- **SELECT** – is used to retrieve data from the a database.

3. **DML(Data Manipulation Language) :** The SQL commands that deals with the manipulation of data present in the database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

Examples of DML:

- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete records from a database table.

4. **DCL(Data Control Language) :** DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

Examples of DCL commands:

- **GRANT**-gives user's access privileges to database.
- **REVOKE**-withdraw user's access privileges given by using the GRANT command.

5. **TCL(transaction Control Language) :** TCL commands deals with the transaction within the database.

Examples of TCL commands:

- **COMMIT**– commits a Transaction.
- **ROLLBACK**– rollbacks a transaction in case of any error occurs.
- **SAVEPOINT**–sets a savepoint within a transaction.
- **SET TRANSACTION**–specify characteristics for the transaction.

1. DDL commands SQL:
create command

Create is a DDL SQL command used to create a table or a database in relational database management system.

Creating a Database

To create a database in RDBMS, **create** command is used. Following is the syntax,

create database <db_name>

Example for creating

Database create database test;

The above command will create a database named **test**, which will be an empty schema without any table.

To create tables in this newly created database, we can again use the create command.

Creating a Table

Create command can also be used to create tables. Now when we create a table, we have to specify the details of the columns of the tables too. We can specify the **names** and **data types** of various columns in the create command itself.

Following is the syntax,

create table <table_name> (
column_name1 datatype1, column_name2 datatype2, column_name3 datatype3, column_name4 datatype4
);

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

Most commonly used data types for Table columns

Datatype	Use
INT	used for columns which will store integer values.
FLOAT	used for columns which will store float values.
DOUBLE	used for columns which will store float values.
VARCHAR	used for columns which will be used to store characters and integers, basically a string.
CHAR	used for columns which will store char values(single character).
DATE	used for columns which will store date values.
TEXT	used for columns which will store text which is generally long in length. For example, if you create a table for storing profile information of a social networking website, then for **about me** section you can have a column of type TEXT.

Example:

SQL> create table bankAccount(id number(3),custname varchar(15),branch varchar(10));

Table created.

SQL> desc bankAccount;

Name	Null?	Type
ID		NUMBER(3)
CUSTNAME		VARCHAR2(15)
BRANCH		VARCHAR2(10)

SQL: ALTER command

alter command is used for altering the table structure, such as,

- to add a column to existing table
- to rename any existing column
- to change datatype of any column or to modify its size.
- to drop a column from the table.

ALTER Command: Add a new Column

Using ALTER command we can add a column to any existing table. Following is the syntax,

ALTER TABLE table_name ADD(column_name datatype);

ALTER TABLE table_name ADD(column_name datatype);

SQL> alter table bankAccount add city varchar(10);

Table altered.

ALTER Command: Add multiple new Columns

Using ALTER command we can even add multiple new columns to any existing table.

Following is the syntax,

ALTER TABLE table_name ADD(column_name1 datatype1, column-name2 datatype2,);

ALTER Command: Add Column with default value

ALTER command can add a new column to an existing table with a default value too. The default value is used when no value is inserted in the column. Following is the syntax,

ALTER TABLE table_name ADD(column-name1 datatype1 DEFAULT some_value);

ALTER Command: Modify an existing Column

ALTERcommand can also be used to modify data type of any existing column. Following is the syntax,

ALTER TABLE table_name modify(column_name

datatype); SQL> alter table bankAccount modify id number(4);

Table altered.

SQL> desc bankAccount;

Name	Null?	Type
ID		NUMBER(4)
CUSTNAME		VARCHAR2(15)
BRANCH		VARCHAR2(10)

ALTER Command: Rename a Column

Using ALTERcommand you can rename an existing column. Following is the syntax,

ALTER TABLE table_name RENAME old_column_name TO new_column_name;

SQL> alter table bankAccount drop column branch;

Table altered.

SQL> desc bankAccount;

Name	Null?	Type
ID		NUMBER(4)
CUSTNAME		VARCHAR2(15)
CITY		VARCHAR2(10)

SQL> alter table bankAccount rename to
acct; Table altered.

SQL> desc acct;

Name	Null?	Type
ID		NUMBER(4)
CUSTNAME		VARCHAR2(15)
CITY		VARCHAR2(10)

ALTER Command: Drop a Column

ALTERcommand can also be used to drop or remove columns. Following is the syntax,

ALTER TABLE table_name DROP(column_name);

TRUNCATE command

TRUNCATE command removes all the records from a table. But this command will not destroy the table's structure. When we use TRUNCATE command on a table its (auto-increment) primary key is also initialized. Following is its syntax,

TRUNCATE TABLE table_name

SQL> truncate table bankAccount;
Table truncated.

SQL> desc bankAccount;

Name	Null?	Type
ID		NUMBER(4)
CUSTNAME		VARCHAR2(15)
CITY		VARCHAR2(10)

DROP command

DROP command completely removes a table from the database. This command will also destroy the table structure and the data stored in it. Following is its syntax,

DROP TABLE table_name

SQL> drop table bankAccount ;

Table dropped.

RENAME query

RENAMEcommand is used to set a new name for any existing table. Following is the syntax,

RENAME TABLE old_table_name to new_table_name

2. DML Command

Using INSERT SQL command

Data Manipulation Language (DML) statements are used for managing data in database. DML commands are not auto-committed. It means changes made by DML command are not permanent to database, it can be rolled back.

INSERT command

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

syntax, INSERT INTO table_name VALUES(data1, data2, ...)

Insert value into only specific columns

We can use the INSERT command to insert values for only some specific columns of a row.

We can specify the column names along with the values to be inserted like this,

INSERT INTO student(id, name) values(value, value);

The above SQL query will only insert id and name values in the newly inserted record.

Insert NULL value to a column

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

SQL> desc acct;

Name	Null?	Type
ID		NUMBER(4)
CUSTNAME		VARCHAR2(15)
CITY		VARCHAR2(10)

INSERT – is used to insert data into a table.

SQL> insert into acct

values(101,'santhosh','mumbai'); 1 row created.

SQL> select * from acct;

ID	CUSTNAME	CITY
101	santhosh	Mumbai

SQL> **insert** into acct

values(&id,'&custname','&city'); Enter value for id:

102

Enter value for custname:

sreeram Enter value for city:

bangalore

old 2: values(&id,'&custname','&city')

new 2: values(102,'sreeram','bangalore')

1 row created.

SQL> /

Enter value for id: 103

Enter value for custname:

mohan Enter value for city:

kerala

old 2: values(&id,'&custname','&city')

new 2: values(103,'mohan','kerala')

1 row
created.

SQL> /

Enter value for id: 104

Enter value for custname: setti

Enter value for city: bengal

old 2: values(&id,'&custname','&city')

new 2: values(104,'setti','bengal')

1 row
created.

SQL> /

Enter value for id: 105

Enter value for custname:

balaji Enter value for city:

delhi

old 2: values(&id,'&custname','&city')

new 2: values(105,'balaji','delhi')

1 row created.

SQL> select * from acct;

ID	CUSTNAME	CITY
101	santhosh	mumbai

102	sreeram	bangalore
103	mohan	kerala
104	setti	bengal
105	balaji	delhi

Using UPDATE SQL command

UPDATE command

UPDATE command is used to update any record of data in a table. Following is its general syntax,

UPDATE table_name SET column_name = new_value WHERE some_condition;

WHERE is used to add a condition to any SQL query, we will soon study about it in detail.

Updating Multiple Columns

We can also update values of multiple columns using a single UPDATE statement.

UPDATE student SET name='Abhi', age=17 where s_id=103;

The above command will update two columns of the record which has s_id 103.

S_ID	NAME	AGE
101	Adam	15
102	Alex	18
103	Abhi	17

UPDATE Command: Incrementing Integer Value

UPDATE student SET age = age+1;

As you can see, we have used age = age + 1 to increment the value of age by 1.

NOTE: This style only works for integer values.

UPDATE – is used to update existing data within a table.

SQL> update acct set custname='raju',city='trichy' where id=104;

1 row updated.

SQL> select * from acct;

ID	CUSTNAME	CITY
101	santhosh	mumbai
102	sreeram	bangalore
103	mohan	kerala
104	raju	trichy
105	balaji	delhi

Using DELETE SQL command

DELETE command

DELETE command is used to delete data from a table. Following is its general syntax,

DELETE FROM table_name;

Let's take a sample table **student**:

S_ID	NAME	AGE
101	Adam	15
102	Alex	18
103	Abhi	17

Delete all Records from a Table

DELETE FROM student;

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

Delete a particular Record from a Table

In our **student** table if we want to delete a single record, we can use the WHERE clause to provide a condition in our DELETE statement.

DELETE FROM student WHERE s_id=103;

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

S_ID	S_NAME	AGE
101	Adam	15
102	Alex	18

Isn't DELETE same as TRUNCATE

TRUNCATE command is different from DELETE command. The delete command will delete all the rows from a table whereas truncate command not only deletes all the records stored in the table, but it also re-initializes the table(like a newly created table).

DELETE – is used to delete records from a database

table. SQL> delete acct where id=103;

1 row deleted.

SQL> select * from acct;

ID	CUSTNAME	CITY
101	santhosh	mumbai
102	sreeram	bangalore
104	raju	trichy
105	balaji	delhi

3. TCL (Transaction Control Language) COMMANDS

COMMIT, ROLLBACK AND SAVEPOINT SQL COMMANDS

Transaction Control Language (TCL) commands is used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements. It also allows statements to be grouped together into logical transactions.

COMMIT command

COMMIT command is used to permanently save any transaction into the database.

To avoid that, we use the COMMIT command to mark the changes as permanent.

Following is commit command's syntax,

COMMIT;

COMMIT– commits a Transaction.

SQL> commit;

Commit complete.

SQL> select * from

acct;

ID	CUSTNAME	CITY
101	santhosh	mumbai
102	sreeram	bangalore
104	raju	trichy
105	balaji	delhi

ROLLBACK command

This command restores the database to last commited state. It is also used with SAVEPOINT command to jump to a savepoint in an ongoing transaction.

Following is rollback command's syntax,

ROLLBACK TO savepoint_name;

SAVEPOINT command

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

Following is savepoint command's syntax,
SAVEPOINT savepoint_name;

SAVEPOINT– sets a savepoint within a transaction.

SQL>savepoint s1;
Savepoint created.

Using Savepoint and Rollback
Following is the table **class**,

ID	NAME
1	Abhi
2	Adam
4	Alex

Let’s use some SQL queries on the above table and see the results.
INSERT INTO class VALUES(5, 'Rahul');

COMMIT;

UPDATE class SET name = 'Abhijit' WHERE id = '5';

SAVEPOINT A;

INSERT INTO class VALUES(6, 'Chris');

SAVEPOINT B;

INSERT INTO class VALUES(7, 'Bravo');

SAVEPOINT C;

SELECT * FROM class;

NOTE: SELECT statement is used to show the data stored in the table.

The resultant table will look like,

ID	NAME
1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris
7	Bravo

Now let's use the ROLLBACK command to roll back the state of data to the **savepoint B**.

ROLLBACK TO B;

ROLLBACK– rollbacks a transaction in case of any error

occurs. SQL> rollback to s1;

Rollback complete.
SELECT * FROM class;
SQL> select * from
acct;

Now our **class** table will look like,

ID	NAME

1	Abhi
2	Adam
4	Alex
5	Abhijit
6	Chris

Now let's again use the ROLLBACK command to roll back the state of data to the **savepoint A**
ROLLBACK TO A;

SELECT * FROM class;
Now the table will look
like,

ID	NAME
1	Abhi
2	Adam
4	Alex
5	Abhijit

So now you know how the commands COMMIT, ROLLBACK and SAVEPOINT works.

ADVANCED SQL FEATURES

Database Querying – Simple Queries, Nested Queries, Sub Queries and Joins

SQL - SELECT Query

The SQL **SELECT** statement is used to fetch the data from a database table which returns this data in the form of a result table. These result tables are called result-sets.

Syntax

SELECT column1, column2, columnN FROM table_name;

Here, column1, column2... are the fields of a table whose values you want to fetch. If you want to fetch all the fields available in the field, then you can use the following syntax.

SELECT * FROM table_name; Example

Consider the CUSTOMERS table having the following records –

ID	NAME	AGE	ADDRESS	SALARY
1.	Ramesh	32	Ahmedabad	2000.00
2.	Khilan	25	| Delhi	1500.00
3.	kaushik	23	Kota	2000.00
4.	Chaitali	25	Mumbai	6500.00
5.	Hardik	27	Bhopal	8500.00
6.	Komal	22	MP	4500.00
7.	Muffy	24	Indore	10000.00

The following code is an example, which would fetch the ID, Name and Salary fields of the customers available in CUSTOMERS table.

sql> select id, name, salary from customers;

This would produce the following result –

ID	NAME	SALARY
1	Ramesh	2000.00
2	Khilan	1500.00
3	kaushik	2000.00
4	Chaitali	6500.00
5	Hardik	8500.00
6	Komal	4500.00
7	Muffy	10000.00

If you want to fetch all the fields of the CUSTOMERS table, then you should use the following query.

sql> select * from customers;

This would produce the result as shown below.

ID	NAME	AGE	ADDRESS	SALARY
1.	Ramesh	32	Ahmedabad	2000.00
2.	Khilan	25	| Delhi	1500.00
3.	kaushik	23	Kota	2000.00
4.	Chaitali	25	Mumbai	6500.00
5.	Hardik	27	Bhopal	8500.00
6.	Komal	22	MP	4500.00
7.	Muffy	24	Indore	10000.00

SUB QUERY IN ORACLE QUERY

While creating a database if we want to extract some information regarding the data in the database then we use a Query.

Example: If we write a simple Query to create a table:

CREATE TABLE Product (Prod_Id Number Not Null, Prod_Name Varchar2(50, Quantity Varchar2(15), Price Number);

Then, the result will be as in the following.

Product Table

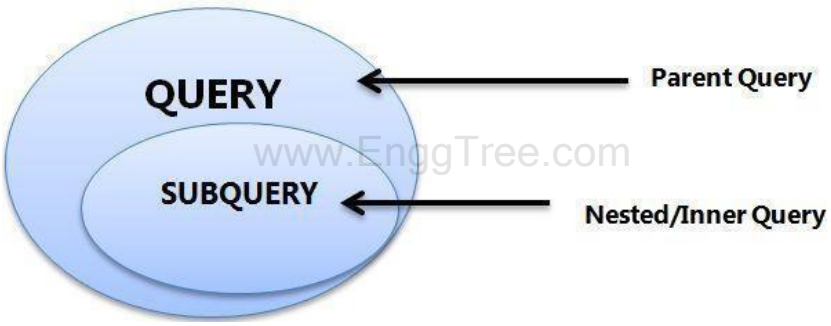
Prod_id	Prod_Name	Quantity	Price

Sub Query

If a Query that contains another Query, then the Query inside the main Query is called a *Sub Query* and the main Query is known as the parent Query. In Oracle the Sub Query will executed on the prior basis and the result will be available to the parent Query and then the execution of the parent/main Query takes place. Sub Queries are very useful for selecting rows from a table having a condition that depends on the data of the table itself. A Sub Query can also be called a Nested/Inner Query.

These Sub Queries can be used with:

- WHERE Clause
- SELECT Clause
- FROM Clause



Syntax

SELECT <column, ...> FROM <table> WHERE expression operator (

SELECT<column,...> FROM<table>WHERE <condition>);

Or

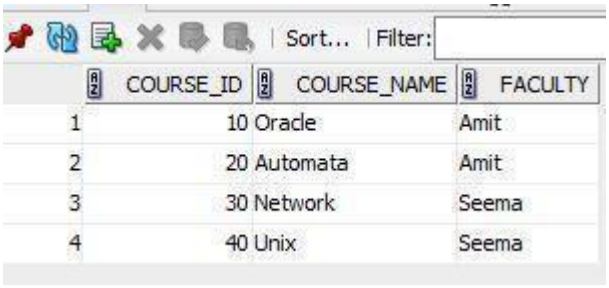
SELECT Col_name [, Col_name] **FROM** table1 [,table2] **WHERE** Col_name OPERATOR (

SELECT Col_name [,Col_name] **FROM** table1 [,table2] [**WHERE**]);

STUDENT TABLE

	STUD_ID	STUD_NAME	AGE	EMAIL	COURSE_ID
1	1	Anjali	26	anjali@a...	10
2	2	Shweta	27	shweta...	30
3	3	Sapna	25	sapna@...	30
4	4	Doli	26	doli@ab...	10

SUBJECT TABLE



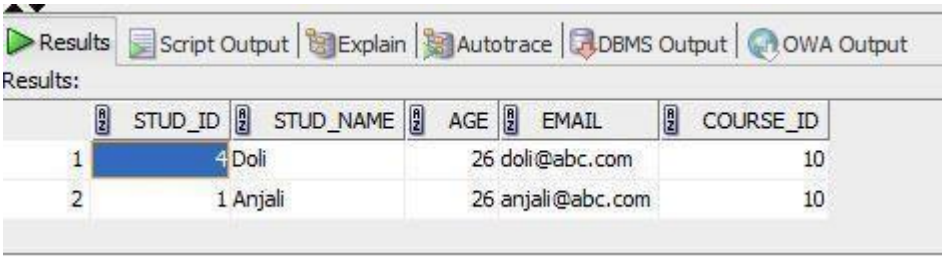
	COURSE_ID	COURSE_NAME	FACULTY
1	10	Oracle	Amit
2	20	Automata	Amit
3	30	Network	Seema
4	40	Unix	Seema

1. **Sub Query using WHERE Clause** SELECT * **FROM** student

WHERE course_id in (**SELECT** course_id

FROM subject

WHERE course_name = 'Oracle')



Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	STUD_ID	STUD_NAME	AGE	EMAIL	COURSE_ID
1	4	Doli	26	doli@abc.com	10
2	1	Anjali	26	anjali@abc.com	10

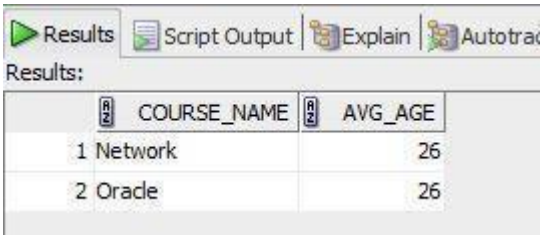
2. Sub Query using FROM Clause

SELECT a.course_name, b.Avg_Age

FROM subject a, (**SELECT** course_id, Avg(Age) **as** Avg_Age

FROM student **GROUP BY** course_id) b

WHERE b.course_id = a.course_id



	COURSE_NAME	AVG_AGE
1	Network	26
2	Oracle	26

3. **Sub Query using SELECT Clause SELECT** course_id, course_name, (

SELECT count (course_id) **as** num_of_student

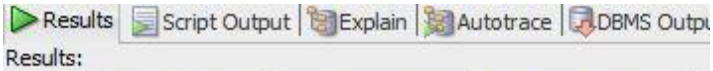
FROM student a

WHERE a.course_id = b.course_id

GROUP BY course_id

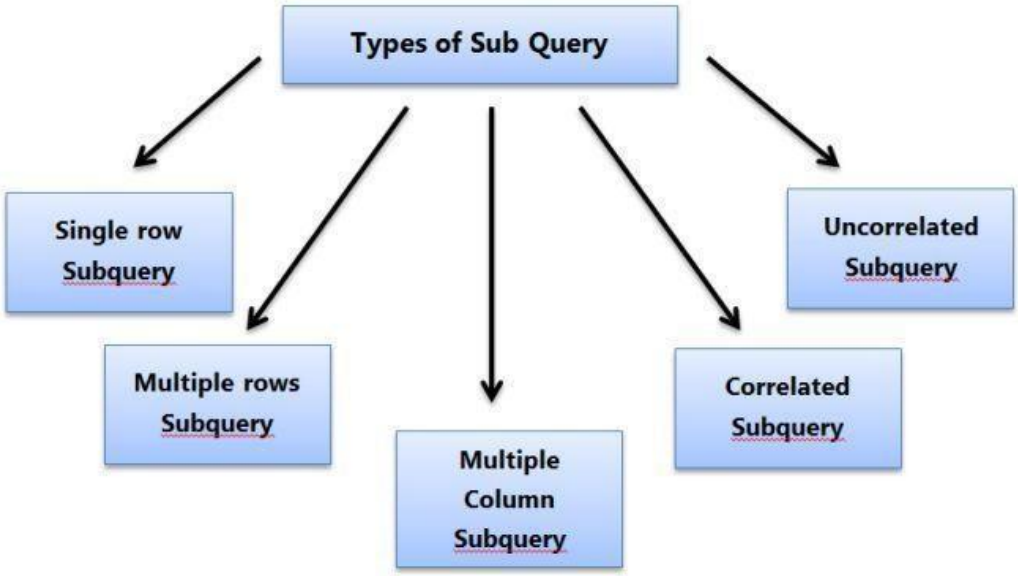
) No_of_Students

FROM subject b



	COURSE_ID	COURSE_NAME	NO_OF_STUDENTS
1	10	Oracle	2
2	20	Automata	(null)
3	30	Network	2
4	40	Unix	(null)

Types of Sub Queries



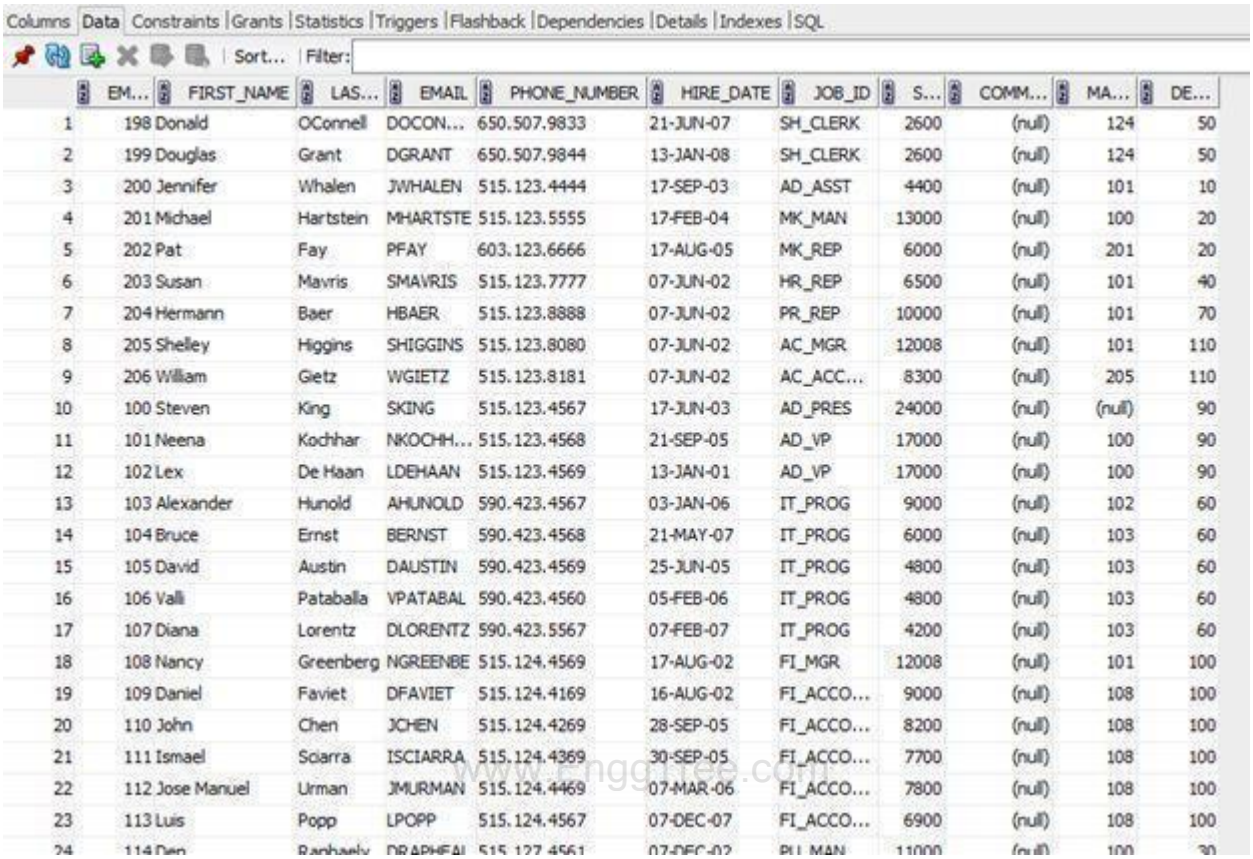
EMPLOYEE TABLE with Column Name

Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Indexes | SQL

Actions...

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
EMPLOYEE_ID	NUMBER(6,0)	No	(null)	1	1	Primary key o...
FIRST_NAME	VARCHAR2(20 BYTE)	Yes	(null)	2		(null) First name of ...
LAST_NAME	VARCHAR2(25 BYTE)	No	(null)	3		(null) Last name of ...
EMAIL	VARCHAR2(25 BYTE)	No	(null)	4		(null) Email id of the...
PHONE_NUMBER	VARCHAR2(20 BYTE)	Yes	(null)	5		(null) Phone numbe...
HIRE_DATE	DATE	No	(null)	6		(null) Date when th...
JOB_ID	VARCHAR2(10 BYTE)	No	(null)	7		(null) Current job of...
SALARY	NUMBER(8,2)	Yes	(null)	8		(null) Monthly salar...
COMMISSION_PCT	NUMBER(2,2)	Yes	(null)	9		(null) Commission p...
MANAGER_ID	NUMBER(6,0)	Yes	(null)	10		(null) Manager id of...
DEPARTMENT_ID	NUMBER(4,0)	Yes	(null)	11		(null) Department id...

EMPLOYEE TABLE with Data

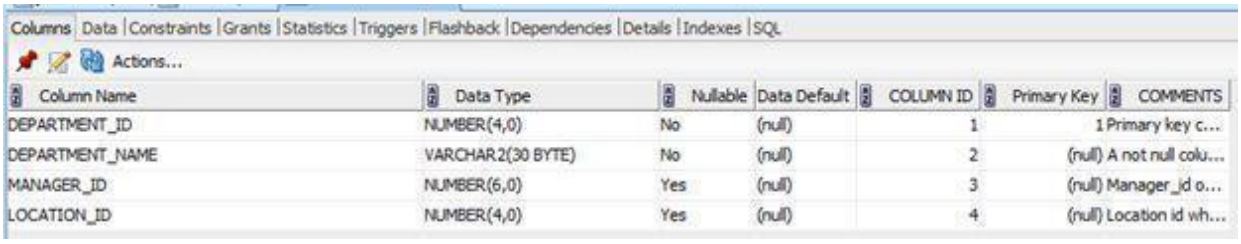


Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Indexes | SQL

Sort... | Filter:

	EM...	FIRST_NAME	LAS...	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	S...	COMM...	MA...	DE...
1	198	Donald	OConnell	DOCON...	650.507.9833	21-JUN-07	SH_CLERK	2600	(null)	124	50
2	199	Douglas	Grant	DGRANT	650.507.9844	13-JAN-08	SH_CLERK	2600	(null)	124	50
3	200	Jennifer	Whalen	JWHALEN	515.123.4444	17-SEP-03	AD_ASST	4400	(null)	101	10
4	201	Michael	Hartstein	MHARTSTE	515.123.5555	17-FEB-04	MK_MAN	13000	(null)	100	20
5	202	Pat	Fay	PFAY	603.123.6666	17-AUG-05	MK_REP	6000	(null)	201	20
6	203	Susan	Mavris	SMAVRIS	515.123.7777	07-JUN-02	HR_REP	6500	(null)	101	40
7	204	Hermann	Baer	HBAER	515.123.8888	07-JUN-02	PR_REP	10000	(null)	101	70
8	205	Shelley	Higgins	SHIGGINS	515.123.8080	07-JUN-02	AC_MGR	12008	(null)	101	110
9	206	William	Gietz	WGIETZ	515.123.8181	07-JUN-02	AC_ACC...	8300	(null)	205	110
10	100	Steven	King	SKING	515.123.4567	17-JUN-03	AD_PRES	24000	(null)	(null)	90
11	101	Neena	Kochhar	NKOCHH...	515.123.4568	21-SEP-05	AD_VP	17000	(null)	100	90
12	102	Lex	De Haan	LDEHAAN	515.123.4569	13-JAN-01	AD_VP	17000	(null)	100	90
13	103	Alexander	Hunold	AHUNOLD	590.423.4567	03-JAN-06	IT_PROG	9000	(null)	102	60
14	104	Bruce	Ernst	BERNST	590.423.4568	21-MAY-07	IT_PROG	6000	(null)	103	60
15	105	David	Austin	DAUSTIN	590.423.4569	25-JUN-05	IT_PROG	4800	(null)	103	60
16	106	Valli	Pataballa	VPATABAL	590.423.4560	05-FEB-06	IT_PROG	4800	(null)	103	60
17	107	Diana	Lorentz	DLORENTZ	590.423.5567	07-FEB-07	IT_PROG	4200	(null)	103	60
18	108	Nancy	Greenberg	NGREENBE	515.124.4569	17-AUG-02	FI_MGR	12008	(null)	101	100
19	109	Daniel	Faviet	DFAVIET	515.124.4169	16-AUG-02	FI_ACCO...	9000	(null)	108	100
20	110	John	Chen	JCHEN	515.124.4269	28-SEP-05	FI_ACCO...	8200	(null)	108	100
21	111	Ismael	Sciarra	ISCIARRA	515.124.4369	30-SEP-05	FI_ACCO...	7700	(null)	108	100
22	112	Jose Manuel	Urman	JMURMAN	515.124.4469	07-MAR-06	FI_ACCO...	7800	(null)	108	100
23	113	Luis	Popp	LPOPP	515.124.4567	07-DEC-07	FI_ACCO...	6900	(null)	108	100
24	114	Den	Raphaely	DRAPHEAL	515.127.4561	07-DEC-02	PU_MAN	11000	(null)	100	30

DEPARTMENT TABLE with Column name

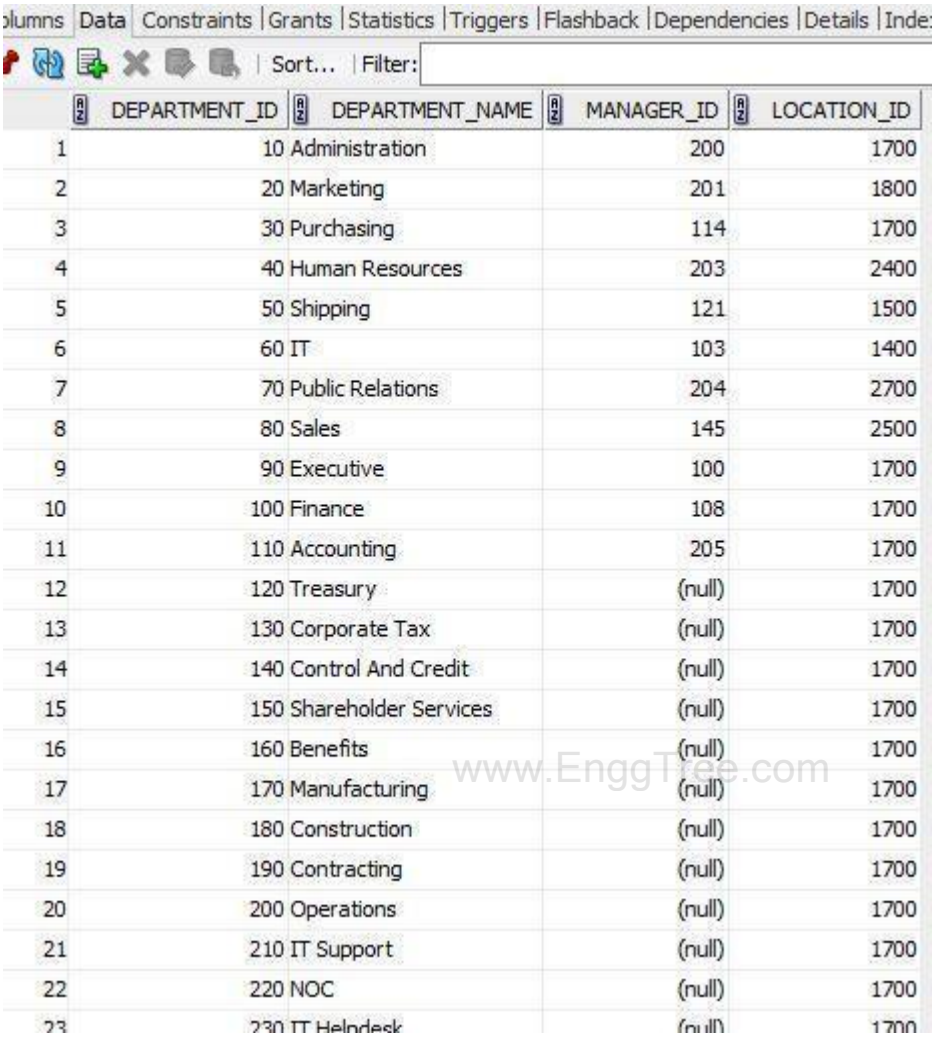


Columns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Indexes | SQL

Actions...

Column Name	Data Type	Nullable	Data Default	COLUMN ID	Primary Key	COMMENTS
DEPARTMENT_ID	NUMBER(4,0)	No	(null)	1	1	Primary key c...
DEPARTMENT_NAME	VARCHAR2(30 BYTE)	No	(null)	2	(null)	A not null colu...
MANAGER_ID	NUMBER(6,0)	Yes	(null)	3	(null)	Manager_id o...
LOCATION_ID	NUMBER(4,0)	Yes	(null)	4	(null)	Location id wh...

DEPARTMENT TABLE with Data



olumns | Data | Constraints | Grants | Statistics | Triggers | Flashback | Dependencies | Details | Inde

Sort... | Filter:

	DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
1	10	Administration	200	1700
2	20	Marketing	201	1800
3	30	Purchasing	114	1700
4	40	Human Resources	203	2400
5	50	Shipping	121	1500
6	60	IT	103	1400
7	70	Public Relations	204	2700
8	80	Sales	145	2500
9	90	Executive	100	1700
10	100	Finance	108	1700
11	110	Accounting	205	1700
12	120	Treasury	(null)	1700
13	130	Corporate Tax	(null)	1700
14	140	Control And Credit	(null)	1700
15	150	Shareholder Services	(null)	1700
16	160	Benefits	(null)	1700
17	170	Manufacturing	(null)	1700
18	180	Construction	(null)	1700
19	190	Contracting	(null)	1700
20	200	Operations	(null)	1700
21	210	IT Support	(null)	1700
22	220	NOC	(null)	1700
23	230	IT Helpdesk	(null)	1700

1. Single Row Sub Query

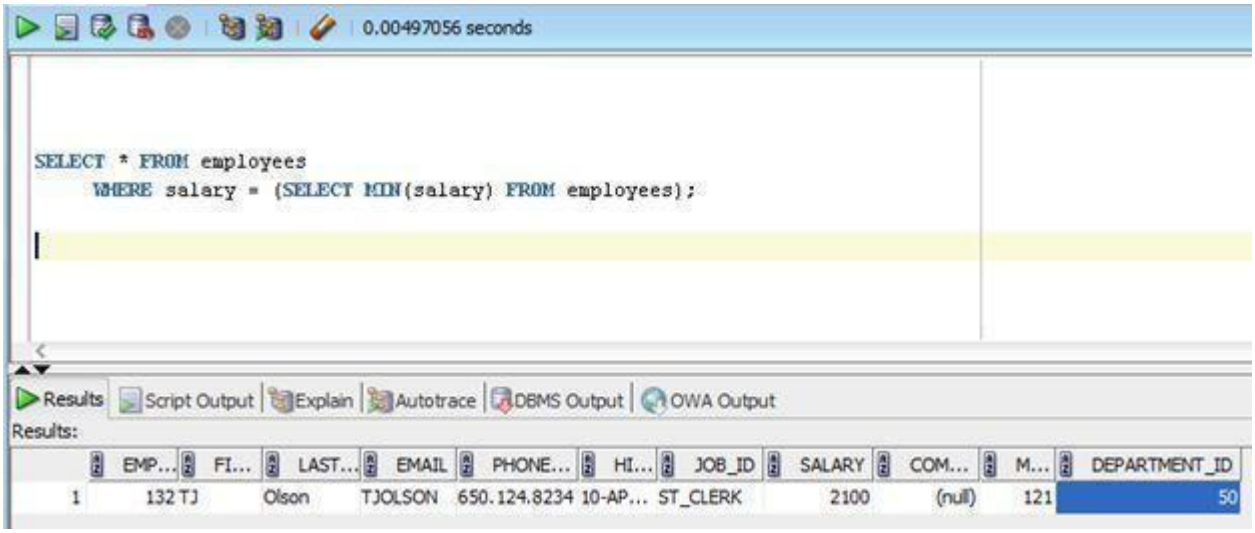
In a Single Row Sub Query the queries return a single/one row of results to the parent/main Query. It can include any of the following operators:

- = Equals to
- Greater than
- < Less than
- >= Greater than Equals to
- <= Less than Equals to

- Not Equals to

Example

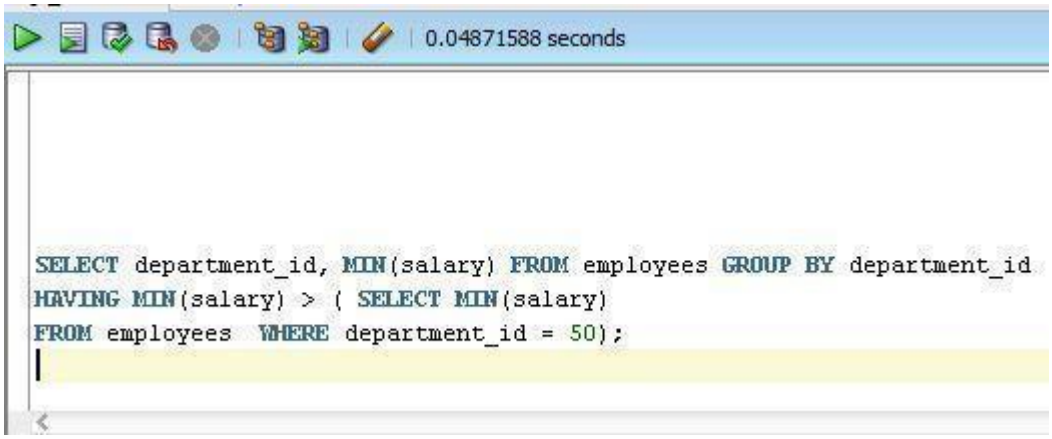
SELECT * FROM employees **WHERE** salary = (**SELECT MIN**(salary) **FROM** employees);

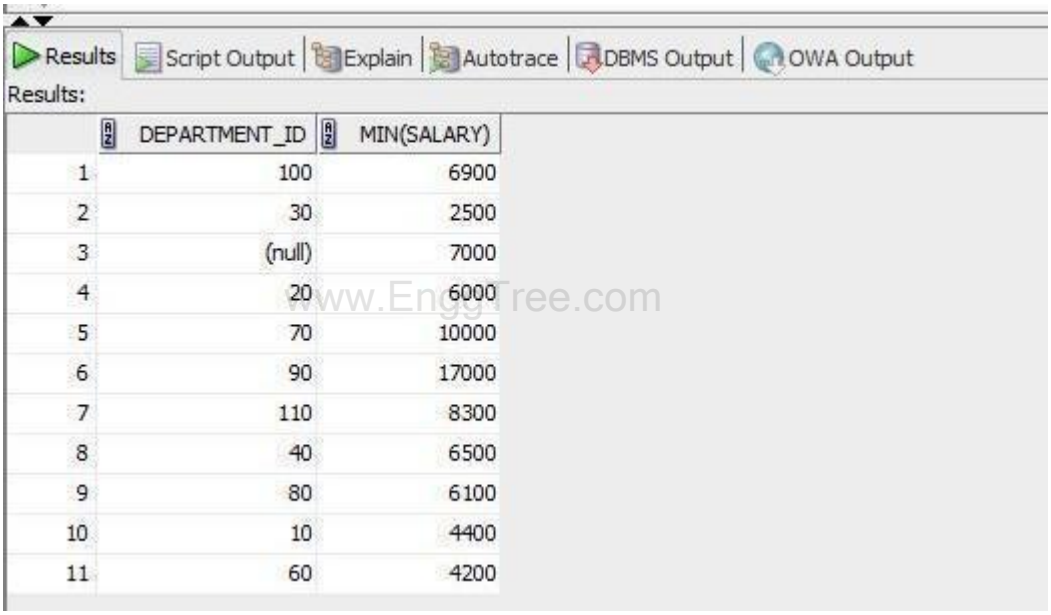


Single Row Sub Query using HAVING Clause

SELECT department_id,
MIN(salary) **FROM** employees
GROUP BY department_id
HAVING MIN(salary) > (**SELECT MIN**(salary)
FROM employees
WHERE department_id = 50);

Execute the Query, the result will be as in the following:





	DEPARTMENT_ID	MIN(SALARY)
1	100	6900
2	30	2500
3	(null)	7000
4	20	6000
5	70	10000
6	90	17000
7	110	8300
8	40	6500
9	80	6100
10	10	4400
11	60	4200

Multiple Row Sub Query

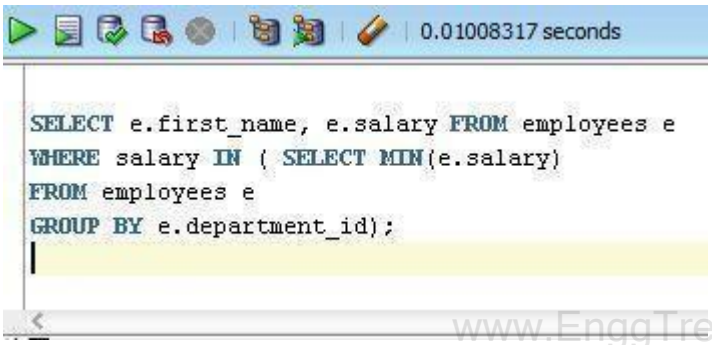
A Multiple Row Sub Query returns a result of multiple rows to the outer/main/parent query. It includes the following operators:

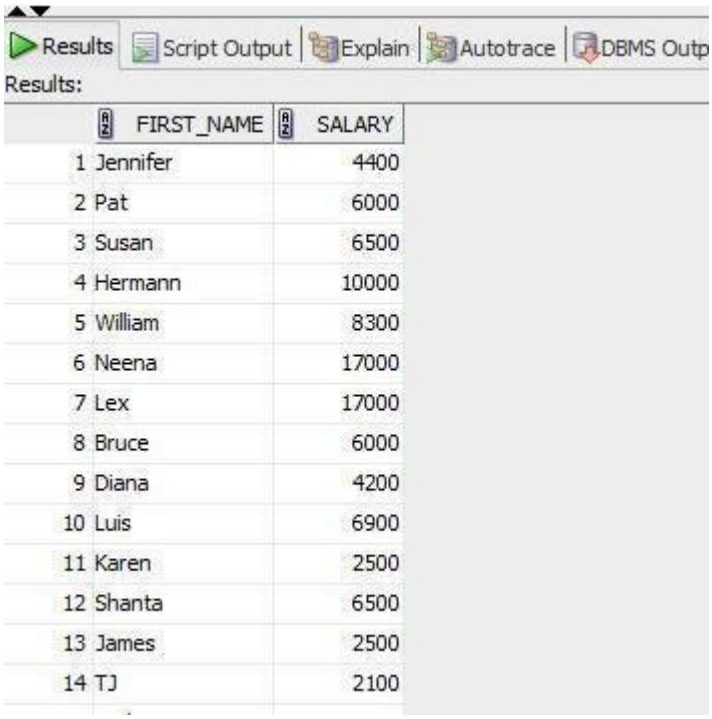
1. IN
2. ANY
3. ALL or EXISTS

Example

```
SELECT e.first_name, e.salary
FROM employees e
WHERE salary IN ( SELECT
MIN(e.salary) FROM employees e
GROUP BY e.department_id);
```

Execute the Query, then the result will be as in the following:





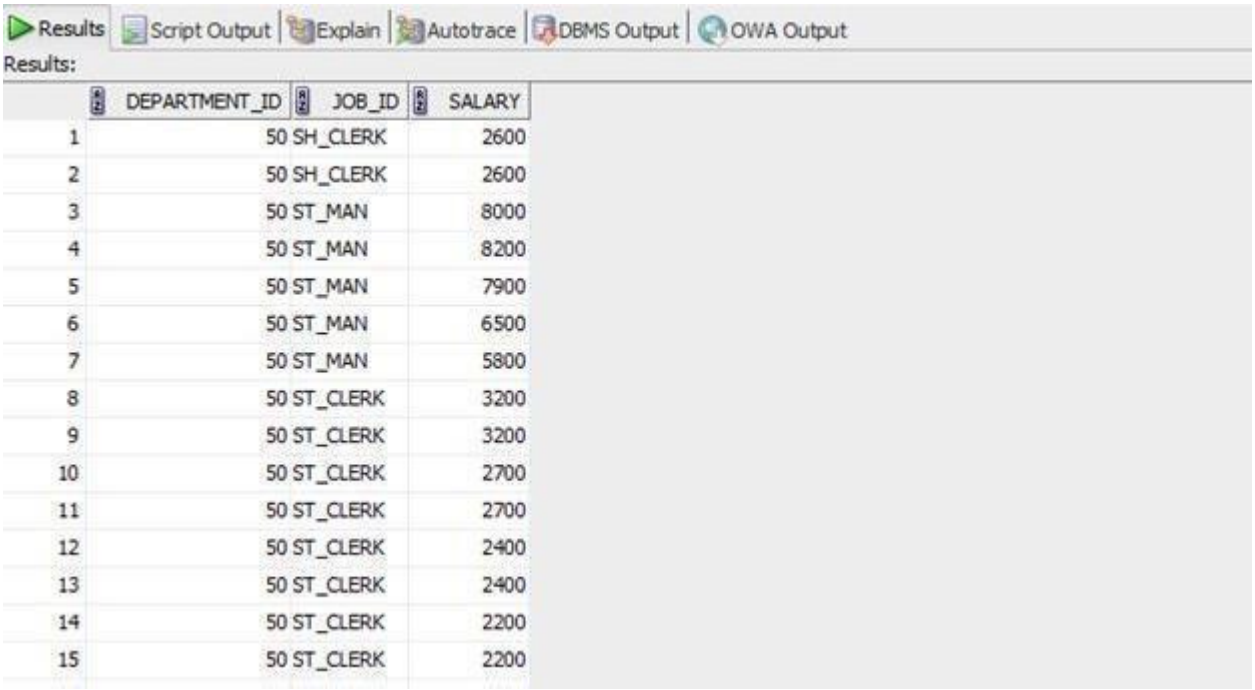
	FIRST_NAME	SALARY
1	Jennifer	4400
2	Pat	6000
3	Susan	6500
4	Hermann	10000
5	William	8300
6	Neena	17000
7	Lex	17000
8	Bruce	6000
9	Diana	4200
10	Luis	6900
11	Karen	2500
12	Shanta	6500
13	James	2500
14	TJ	2100

Multiple Column Sub Query

Multiple Column Sub Queries are queries that return multiple columns to the outer SQL query. It uses the IN operator for the WHERE and HAVING clause.

SELECT e.department_id, e.job_id,e.salary
FROM employees e
WHERE (e.job_id, e.salary) IN (**SELECT** e.job_id, e.salary
FROM employees e
WHERE e.department_id = 50) ;

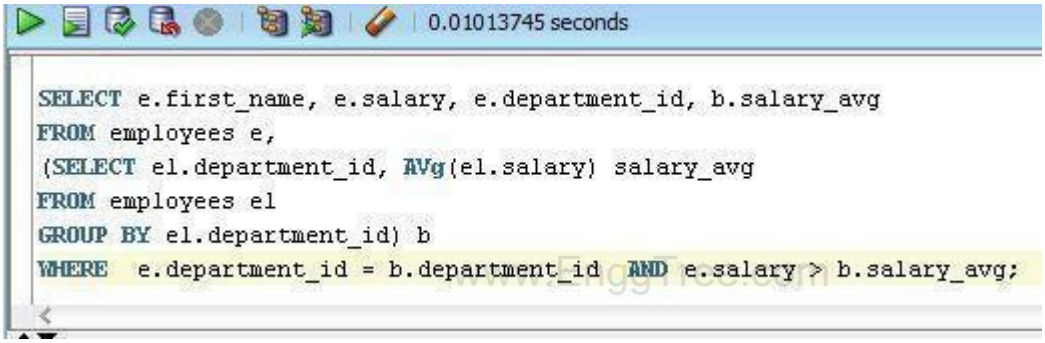
Execute the Query, then the result will be as in the following:

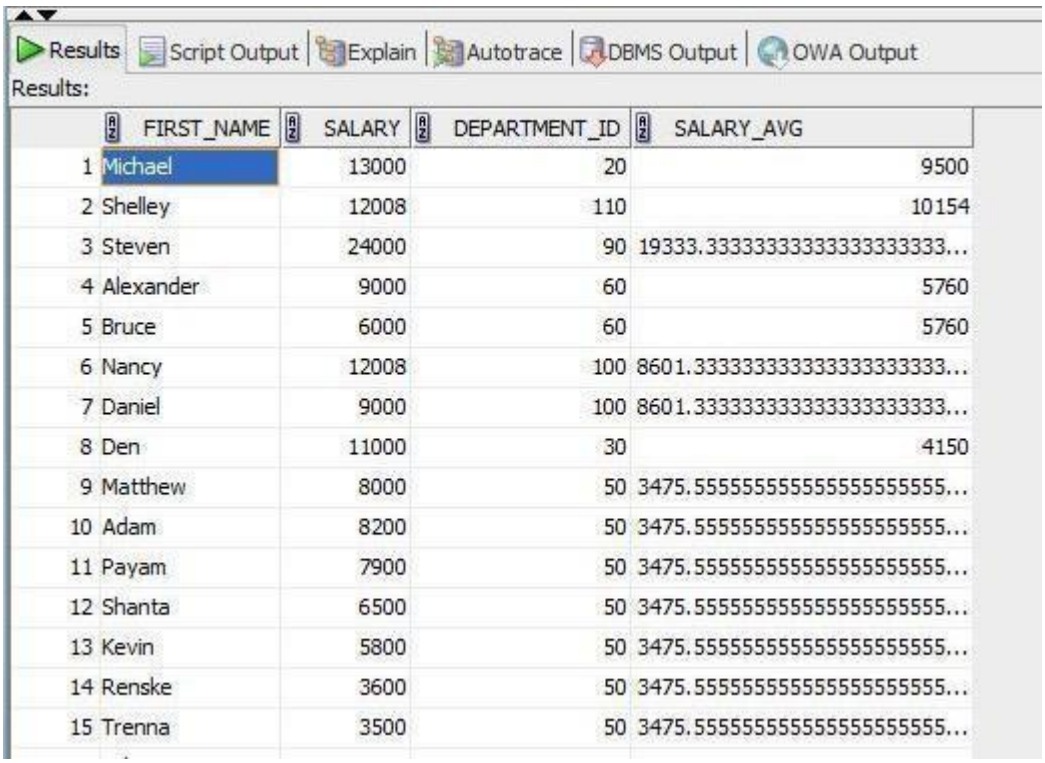


	DEPARTMENT_ID	JOB_ID	SALARY
1	50	SH_CLERK	2600
2	50	SH_CLERK	2600
3	50	ST_MAN	8000
4	50	ST_MAN	8200
5	50	ST_MAN	7900
6	50	ST_MAN	6500
7	50	ST_MAN	5800
8	50	ST_CLERK	3200
9	50	ST_CLERK	3200
10	50	ST_CLERK	2700
11	50	ST_CLERK	2700
12	50	ST_CLERK	2400
13	50	ST_CLERK	2400
14	50	ST_CLERK	2200
15	50	ST_CLERK	2200

Note: We can use a Sub Query using a FROM clause in the main query.

SELECT e.first_name, e.salary, e.department_id, b.salary_avg
FROM employees e,
(**SELECT** e1.department_id, AVg(e1.salary) salary_avg
FROM employees e1
GROUP BY e1.department_id) b
WHERE e.department_id = b.department_id AND e.salary > b.salary_avg; Execute the Query, then the result will be as in the following:





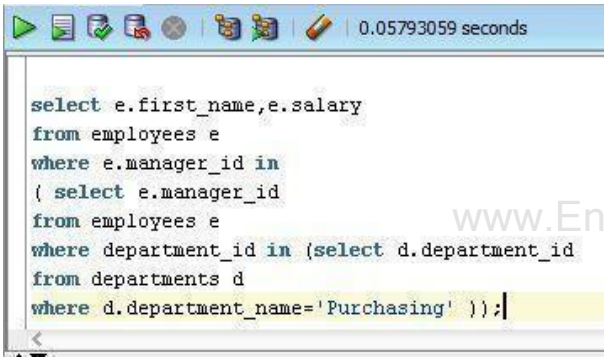
	FIRST_NAME	SALARY	DEPARTMENT_ID	SALARY_AVG
1	Michael	13000	20	9500
2	Shelley	12008	110	10154
3	Steven	24000	90	19333.333333333333333333...
4	Alexander	9000	60	5760
5	Bruce	6000	60	5760
6	Nancy	12008	100	8601.333333333333333333...
7	Daniel	9000	100	8601.333333333333333333...
8	Den	11000	30	4150
9	Matthew	8000	50	3475.555555555555555555...
10	Adam	8200	50	3475.555555555555555555...
11	Payam	7900	50	3475.555555555555555555...
12	Shanta	6500	50	3475.555555555555555555...
13	Kevin	5800	50	3475.555555555555555555...
14	Renske	3600	50	3475.555555555555555555...
15	Trenna	3500	50	3475.555555555555555555...

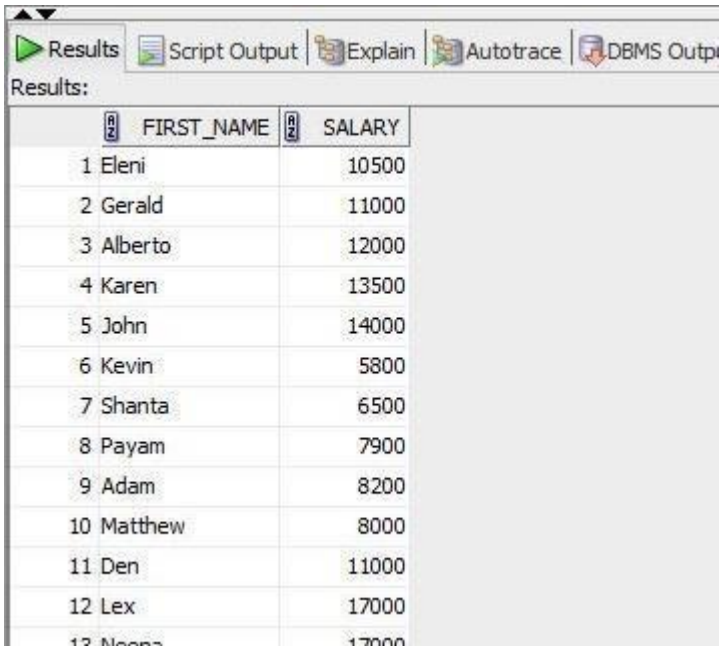
Nested Sub Query

When we write a Sub Query in a WHERE and HAVING clause of another Sub Query then it is called a nested Sub Query.

SELECT e.first_name,e.salary
FROM employees e **WHERE** e.manager_id in (**SELECT** e.manager_id **FROM** employees e
WHERE department_id in (**select** d.department_id
FROM departments d
WHERE d.department_name='Purchasing'));

Execute the Query, then the result will be as in the following:





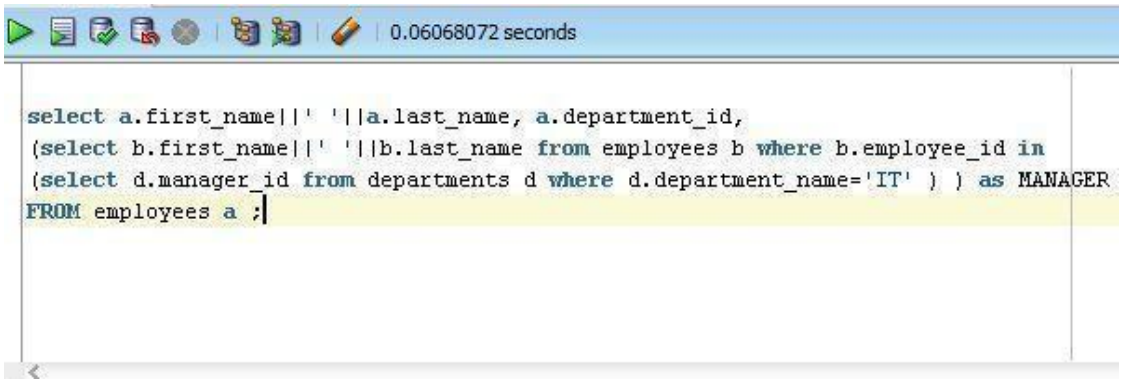
	FIRST_NAME	SALARY
1	Eleni	10500
2	Gerald	11000
3	Alberto	12000
4	Karen	13500
5	John	14000
6	Kevin	5800
7	Shanta	6500
8	Payam	7900
9	Adam	8200
10	Matthew	8000
11	Den	11000
12	Lex	17000

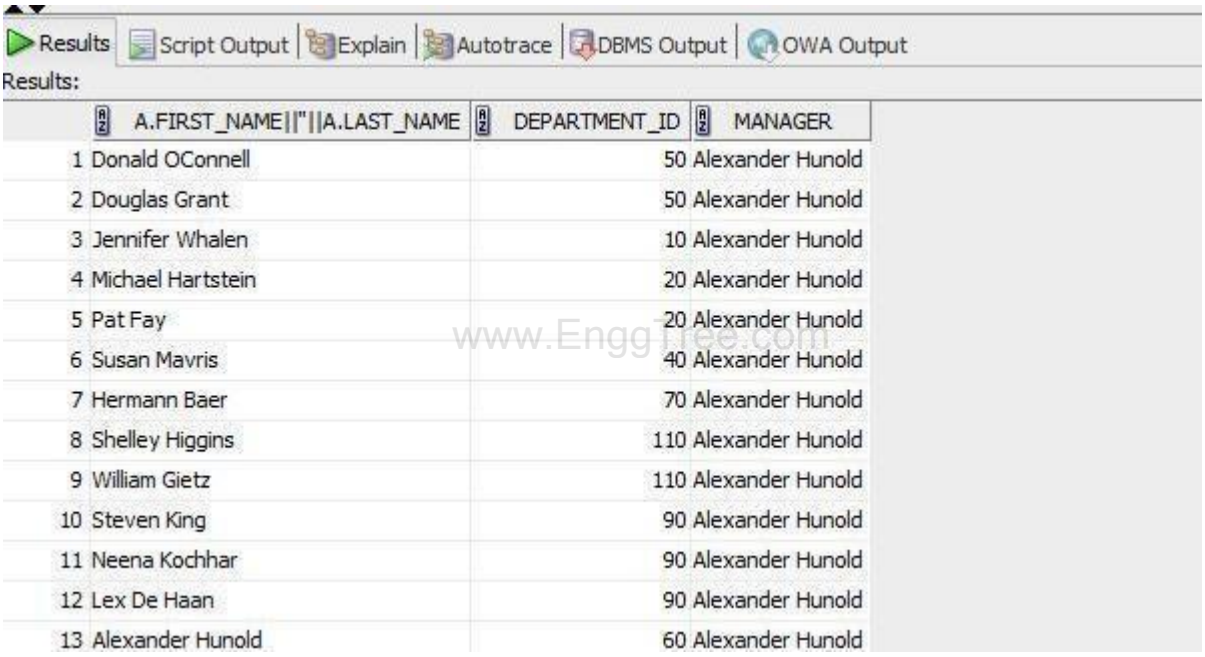
Correlated Sub Query

A Correlated Sub Query contains a reference to a table that appears in the outer query. It is used for row by row processing, in other words the Sub Query will execute row by row for the parent query.

```
SELECT a.first_name||' '||a.last_name, a.department_id, (SELECT b.first_name||' '||b.last_name
FROM employees b WHERE b.employee_id in (SELECT d.manager_id FROM departments d
WHERE d.department_name='IT' ) ) as MANAGER
FROM employees a ;
```

Execute the Query, then the result will be as in the following:





	A.FIRST_NAME||" "||A.LAST_NAME	DEPARTMENT_ID	MANAGER
1	Donald OConnell	50	Alexander Hunold
2	Douglas Grant	50	Alexander Hunold
3	Jennifer Whalen	10	Alexander Hunold
4	Michael Hartstein	20	Alexander Hunold
5	Pat Fay	20	Alexander Hunold
6	Susan Mavris	40	Alexander Hunold
7	Hermann Baer	70	Alexander Hunold
8	Shelley Higgins	110	Alexander Hunold
9	William Gietz	110	Alexander Hunold
10	Steven King	90	Alexander Hunold
11	Neena Kochhar	90	Alexander Hunold
12	Lex De Haan	90	Alexander Hunold
13	Alexander Hunold	60	Alexander Hunold

DBMS | Nested Queries in SQL

In nested queries, a query is written inside a query. The result of inner query is used in execution of outer query. We will use **STUDENT, COURSE, STUDENT_COURSE** tables for understanding nested queries.

STUDENT

S_ID	S_NAME	S_ADDRESS	S_PHONE	S_AGE
S1	RAM	DELHI	9455123451	18
S2	RAMESH	GURGAON	9652431543	18
S3	SUJIT	ROHTAK	9156253131	20
S4	SURESH	DELHI	9156768971	18

COURSE

C_ID	C_NAME
C1	DSA
C2	Programming
C3	DBMS

STUDENT_COURSE

S_ID	C_ID
S1	C1
S1	C3
S2	C1
S3	C2
S4	C2
S4	C3

There are mainly two types of nested queries:

Independent Nested Queries: In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc. are used in writing independent nested queries.

IN: If we want to find out **S_ID** who are enrolled in **C_NAME** 'DSA' or 'DBMS', we can write it with the help of independent nested query and IN operator. From **COURSE** table, we

can find out **C_ID** for **C_NAME** 'DSA' or DBMS' and we can use these **C_ID**s for finding **S_ID**s from **STUDENT_COURSE** TABLE.

STEP 1: Finding **C_ID** for **C_NAME** ='DSA' or 'DBMS'
Select **C_ID** from **COURSE** where **C_NAME** = 'DSA' or **C_NAME** = 'DBMS'

STEP 2: Using **C_ID** of step 1 for finding **S_ID**
Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME** = 'DSA' or **C_NAME**='DBMS');
The inner query will return a set with members C1 and C3 and outer query will return those **S_ID**s for which **C_ID** is equal to any member of set (C1 and C3 in this case). So, it will return S1, S2 and S4.

Note: If we want to find out names of **STUDENT**s who have either enrolled in 'DSA' or 'DBMS', it can be done as:

Select S_NAME from **STUDENT** where **S_ID** IN
(Select **S_ID** from **STUDENT_COURSE** where **C_ID** IN

(SELECT **C_ID** from **COURSE** where **C_NAME**='DSA' or **C_NAME**='DBMS'));

NOT IN: If we want to find out **S_ID**s of **STUDENT**s who have neither enrolled in 'DSA' nor in 'DBMS', it can be done as:

Select **S_ID** from **STUDENT** where **S_ID** NOT IN

(Select **S_ID** from **STUDENT_COURSE** where **C_ID**

IN

(SELECT **C_ID** from **COURSE** where **C_NAME**='DSA' or **C_NAME**='DBMS'));

The innermost query will return a set with members C1 and C3. Second inner query will return those **S_ID**s for which **C_ID** is equal to any member of set (C1 and C3 in this case) which are S1, S2 and S4. The outermost query will return those **S_ID**s where **S_ID** is not a member of set
(S1, S2 and S4). So it will return S3.

Co-related Nested Queries: In co-related nested queries, the output of inner query depends on the row which is being currently executed in outer query. e.g.; If we want to find out **S_NAME** of **STUDENT**s who are enrolled in **C_ID** 'C1', it can be done with the help of co- related nested query as:

Select S_NAME from **STUDENT** S where EXISTS (select * from **STUDENT_COURSE** SC
where S.**S_ID**=SC.**S_ID** and SC.**C_ID**='C1');

For each row of **STUDENT** S, it will find the rows from **STUDENT_COURSE** where S.**S_ID**
= SC.**S_ID** and SC.**C_ID**='C1'. If for a **S_ID** from **STUDENT** S, atleast a row exists in **STUDENT_COURSE** SC with **C_ID**='C1', then inner query will return true and corresponding **S_ID** will be returned as output.

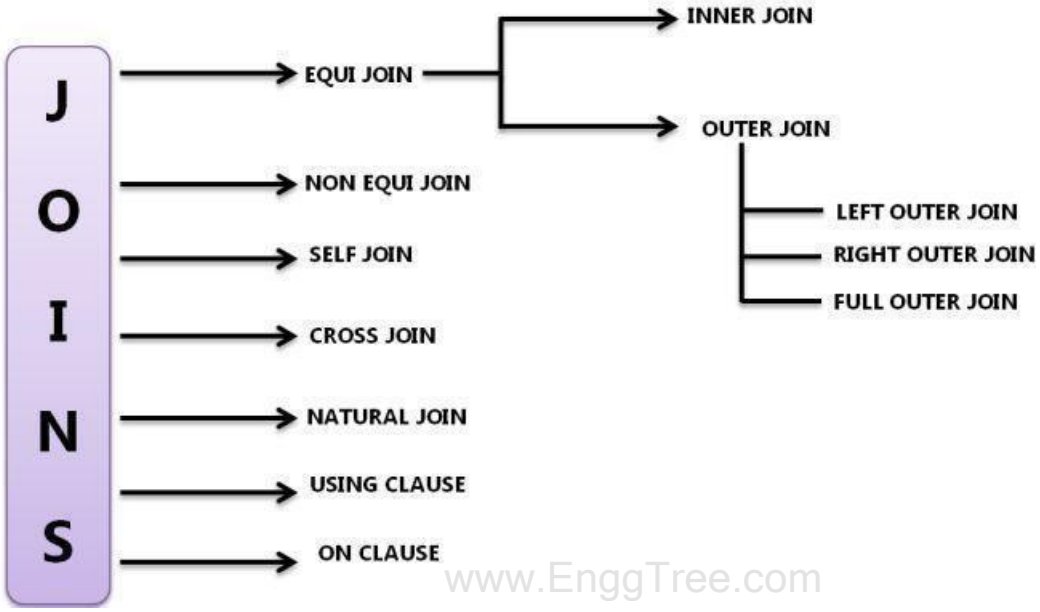
JOINS IN ORACLE

In Oracle, a join is the most powerful operation for merging information from multiple tables based on a common field. There are various types of joins but an INNER JOIN is the common of them.

Syntax

SELECT col1, col2, col3...
FROM table_name1,
table_name2
WHERE table_name1.col2 = table_name2.col1;

Types Of Joins



To understand each of the preceding joins clearly we are assuming the following "CUSTOMER" and "ORDERS" tables: CREATE TABLE Customer

(

Cust_id Number(10) NOT NULL, Cust_name

varchar2(20), Country varchar2(20), Receipt_no

Number(10),

Order_id Number(10) NOT NULL,

);

CREATE TABLE Orders

(

Order_id Number(10), Item_ordered varchar2(20), Order_date date

);

Table: CUSTOMER

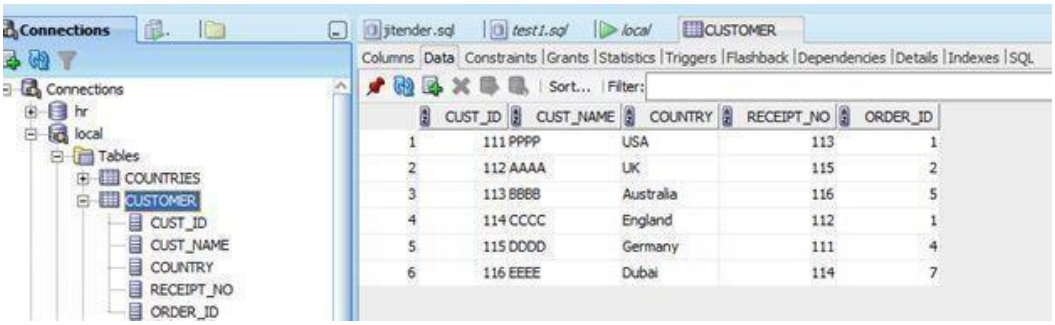
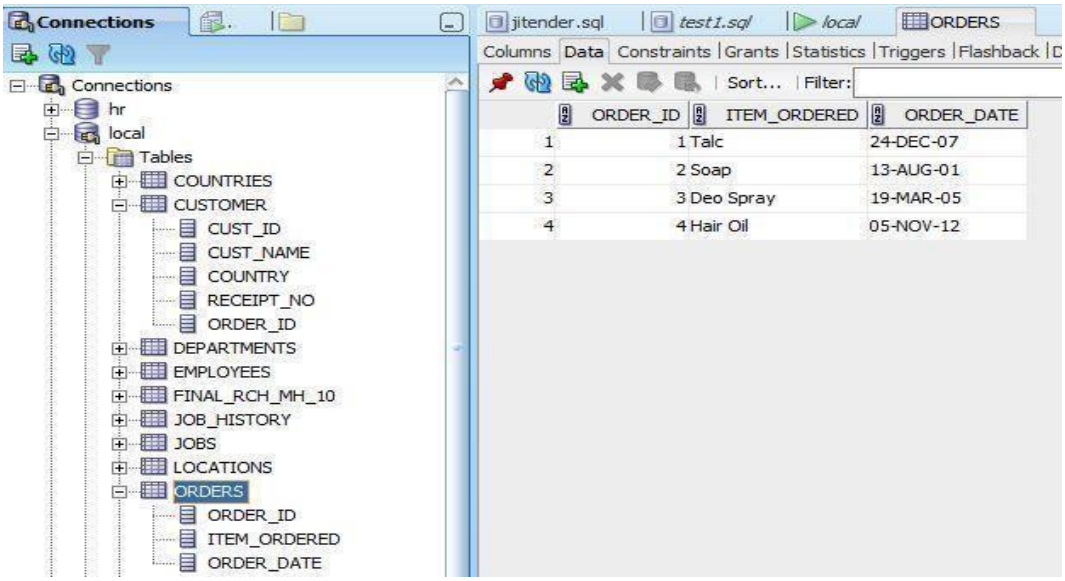


Table: ORDERS



First of all we will explain the "USING" clause and the "ON" clause.

1. Using Clause

To join a table using the USING Clause we write the following command.

Query

SELECT Cust_id, Cust_name, Country, item_Ordered, Order_date

FROM Customer C JOIN Orders O USING (Order_id);

Execution of the query with result

```
SELECT cust_id, cust_name, country, Item_ordered, order_date
FROM customer c join orders o
USING (order_id);
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	CUST_ID	CUST_NAME	COUNTRY	ITEM_ORDERED	ORDER_DAT
1	111	PPPP	USA	Talc	24-DEC-07
2	114	CCCC	England	Talc	24-DEC-07
3	112	AAAA	UK	Soap	13-AUG-01
4	115	DDDD	Germany	Hair Oil	05-NOV-12

2. On Clause

To join a table using an ON Clause we write the following command:

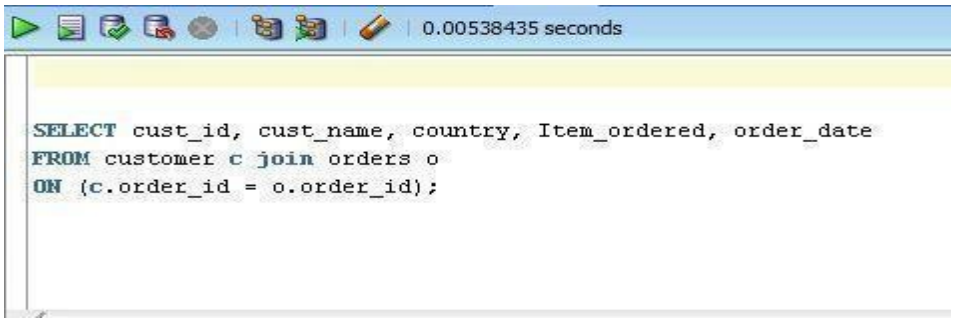
Query

SELECT Cust_id, Cust_name, Country, item_Ordered, Order_date

FROM Customer C JOIN Orders O USING (C.Order_id =

O.Order_id);

Execution of the query with result



	CUST_ID	CUST_NAME	COUNTRY	ITEM_ORDERED	ORDER_DATE
1	111	PPPP	USA	Talc	24-DEC-07
2	114	CCCC	England	Talc	24-DEC-07
3	112	AAAA	UK	Soap	13-AUG-01
4	115	DDDD	Germany	Hair Oil	05-NOV-12

Equi Join

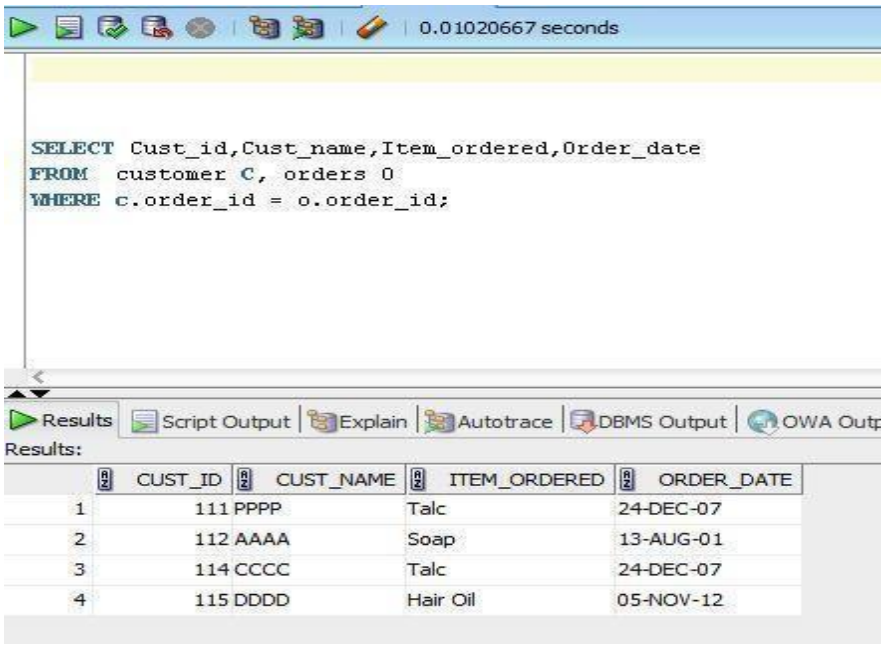
An Equi join is used to get the data from multiple tables where the names are common and the columns are specified. It includes the equal ("=") operator.

Example

SELECT Cust_id, Cust_name, item_Ordered, Order_date

FROM Customer C, Orders O WHERE C.Order_id = O.Order_id;

Execution of the query with result



1. Inner Join

An Inner Join retrieves the matching records, in other words it retrieves all the rows where there is at least one match in the tables.

Example

SELECT Cust_id, Cust_name, Country, item_ordered, Order_date

FROM Customer INNER JOIN Orders USING (Order_id);

Execution of the query with result

```
SELECT cust_id, cust_name, country, Item_ordered, order_date
FROM customer INNER JOIN orders
USING (Order_id);
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	CUST_ID	CUST_NAME	COUNTRY	ITEM_ORDERED	ORDER_DATE
1	111	PPPP	USA	Talc	24-DEC-07
2	114	CCCC	England	Talc	24-DEC-07
3	112	AAAA	UK	Soap	13-AUG-01
4	115	DDDD	Germany	Hair Oil	05-NOV-12

2. Outer Join

The records that don't match will be retrieved by the Outer join. It is of the following three types:

1. Left Outer Join
2. Right Outer Join
3. Full Outer Join

1. Left Outer Join

A Left outer join retrieves all records from the left hand side of the table with all the matched records. This query can be written in one of the following two ways.

Example

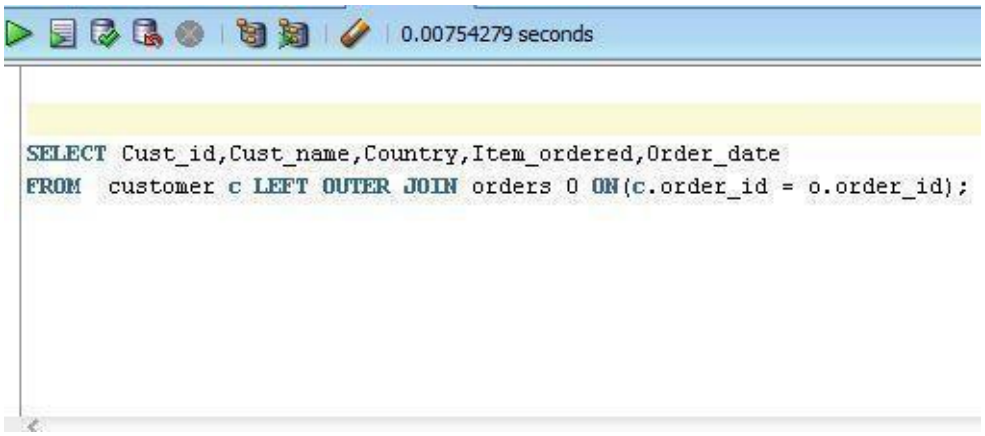
Method

1

SELECT Cust_id, Cust_name, Country, item_ordered, Order_date

FROM customer C, LEFT OUTER JOIN Orders O ON (C. Order_id = O.Order_id)

Execution of the query with result



	CUST_ID	CUST_NAME	COUNTRY	ITEM_ORDERED	ORDER_DATE
1	111	PPPP	USA	Talc	24-DEC-07
2	112	AAAA	UK	Soap	13-AUG-01
3	113	BBBB	Australia	(null)	(null)
4	114	CCCC	England	Talc	24-DEC-07
5	115	DDDD	Germany	Hair Oil	05-NOV-12
6	116	EEEE	Dubai	(null)	(null)

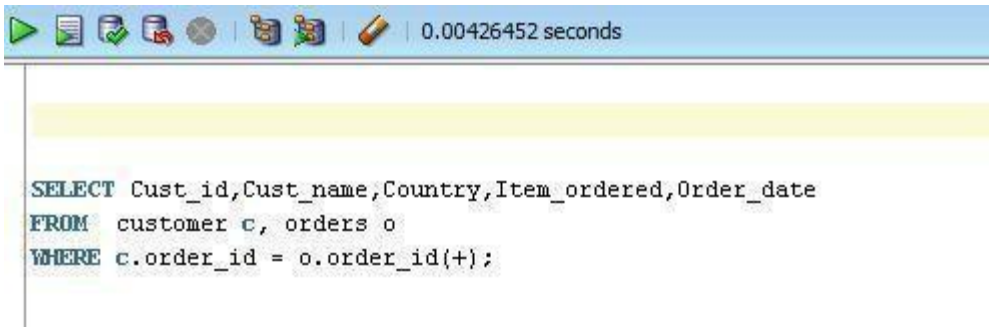
Or: Method 2

SELECT Cust_id, Cust_name, Country, item_ordered, Order_date

FROM customer C, Orders O

WHERE C.Order_id = O.Order_id(+);

Execution of the query with result



	CUST_ID	CUST_NAME	COUNTRY	ITEM_ORDERED	ORDER_DATE
1	111	PPPP	USA	Talc	24-DEC-07
2	112	AAAA	UK	Soap	13-AUG-01
3	113	BBBB	Australia	(null)	(null)
4	114	CCCC	England	Talc	24-DEC-07
5	115	DDDD	Germany	Hair Oil	05-NOV-12
6	116	EEEE	Dubai	(null)	(null)

2. Right Outer Join

A Right Outer Join retrieves the records from the right hand side columns.

Example

Method

1

SELECT Cust_id, Cust_name, Country, item_ordered, Order_date

FROM customer C, RIGHT OUTER JOIN Orders O ON (C. Order_id = O.Order_id)

Execution of the query with result

```
SELECT Cust_id,Cust_name,Country,Item_ordered,Order_date
FROM  customer c RIGHT OUTER JOIN orders O ON(c.order_id = o.order_id);
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

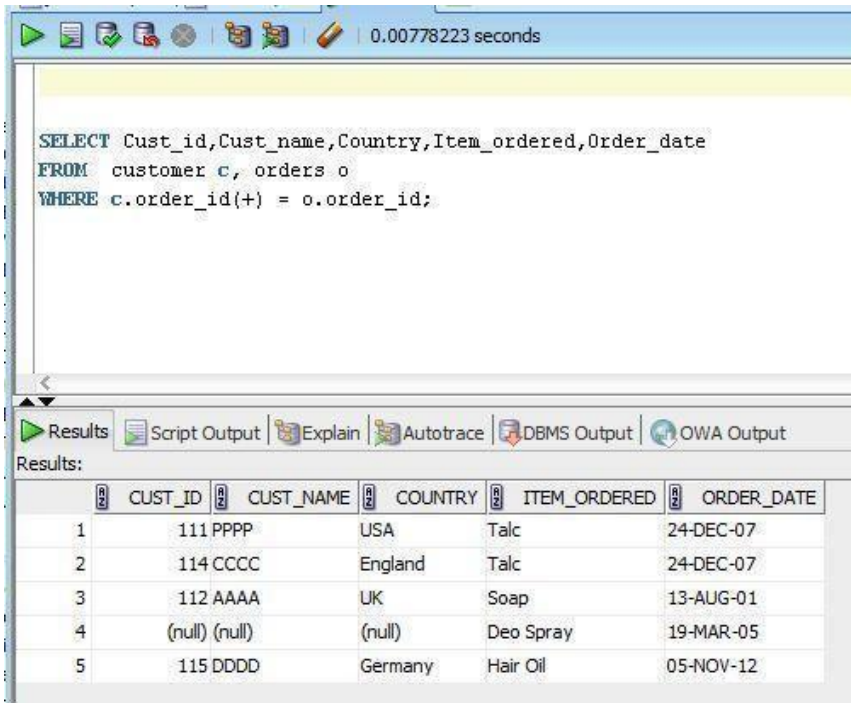
Results:

	CUST_ID	CUST_NAME	COUNTRY	ITEM_ORDERED	ORDER_DATE
1	111	PPPP	USA	Talc	24-DEC-07
2	114	CCCC	England	Talc	24-DEC-07
3	112	AAAA	UK	Soap	13-AUG-01
4	(null)	(null)	(null)	Deo Spray	19-MAR-05
5	115	DDDD	Germany	Hair Oil	05-NOV-12

Or: Method 2

SELECT Cust_id, Cust_name, Country, item_ordered, Order_date
FROM customer C, Orders O WHERE C.Order_id(+)=
O.Order_id;

Execution of the query with result



	CUST_ID	CUST_NAME	COUNTRY	ITEM_ORDERED	ORDER_DATE
1	111	PPPP	USA	Talc	24-DEC-07
2	114	CCCC	England	Talc	24-DEC-07
3	112	AAAA	UK	Soap	13-AUG-01
4	(null)	(null)	(null)	Deo Spray	19-MAR-05
5	115	DDDD	Germany	Hair Oil	05-NOV-12

3. Full Outer Join

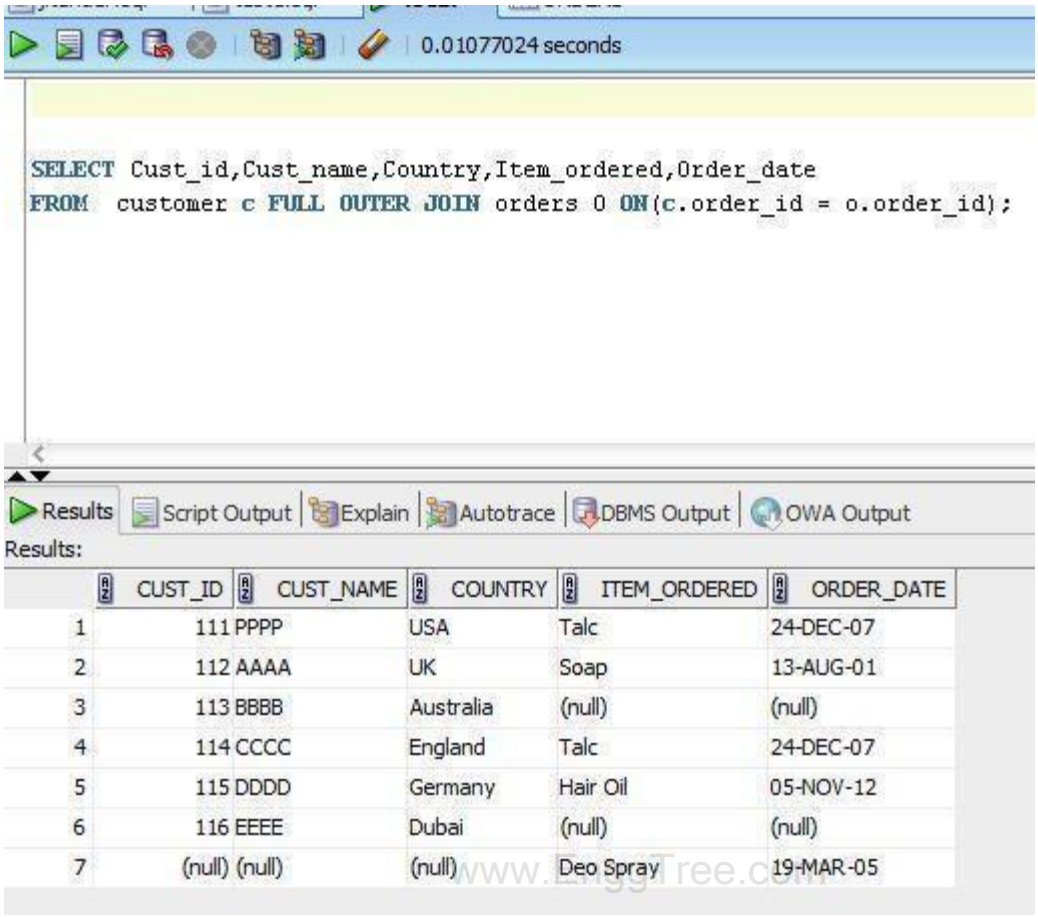
To retrieve all the records, both matching and unmatched from all the tables then use the FULL OUTER JOIN.

Example

SELECT Cust_id, Cust_name, Country, item_ordered, Order_date

FROM customer C, FULL OUTER JOIN Orders OON (C. Order_id = O.Order_id)

Execution of the query with result



2. Non-Equi Join

A Non-Equi join is based on a condition using an operator other than equal to "=".

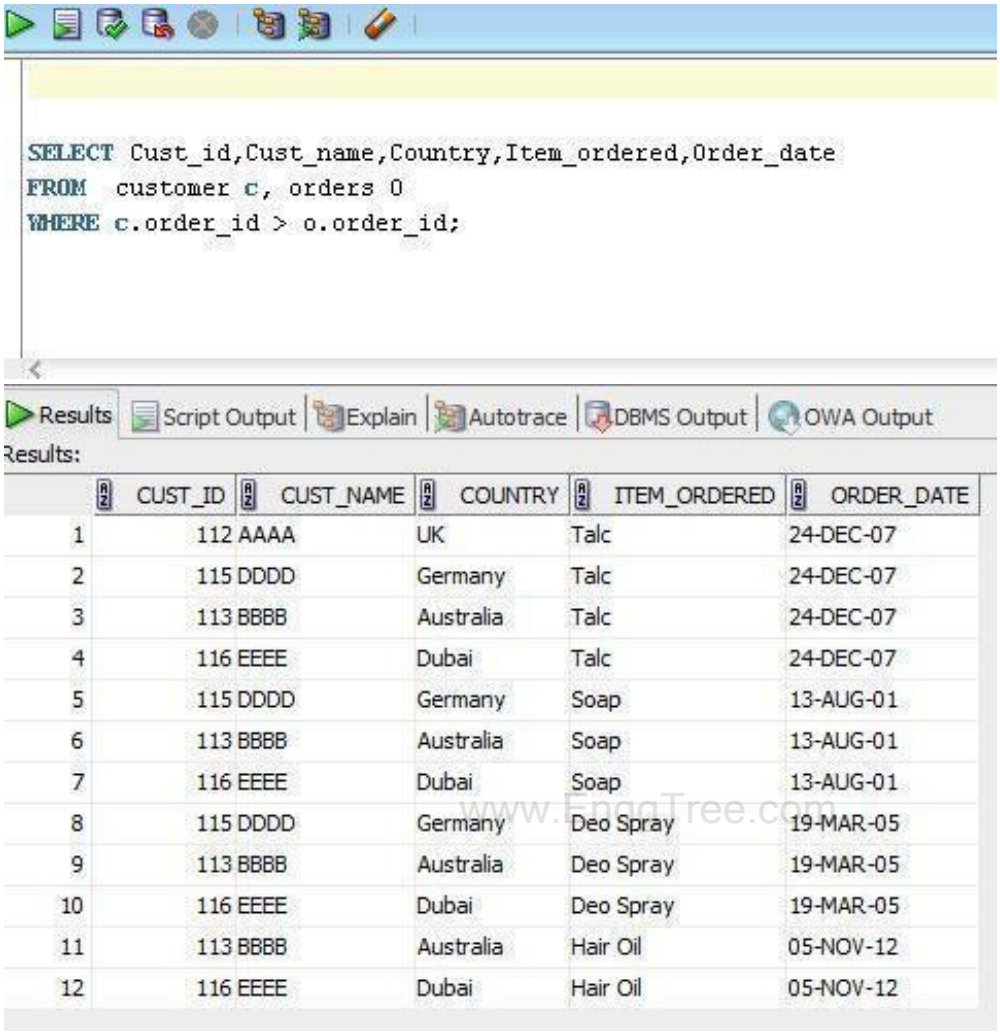
Example

SELECT Cust_id, Cust_name, Country, Item_ordered, Order_date

FROM Customer C, Oredrs O WHERE C. Order_id >

O.Order_id;

Execution of the query with result:

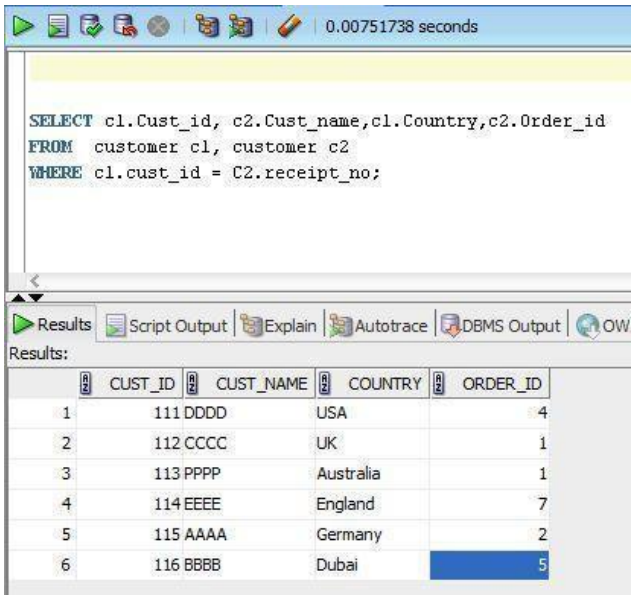


```
SELECT Cust_id,Cust_name,Country,Item_ordered,Order_date
FROM  customer c, orders O
WHERE c.order_id > o.order_id;
```

Results | Script Output | Explain | Autotrace | DBMS Output | OWA Output

Results:

	CUST_ID	CUST_NAME	COUNTRY	ITEM_ORDERED	ORDER_DATE
1	112	AAAA	UK	Talc	24-DEC-07
2	115	DDDD	Germany	Talc	24-DEC-07
3	113	BBBB	Australia	Talc	24-DEC-07
4	116	EEEE	Dubai	Talc	24-DEC-07
5	115	DDDD	Germany	Soap	13-AUG-01
6	113	BBBB	Australia	Soap	13-AUG-01
7	116	EEEE	Dubai	Soap	13-AUG-01
8	115	DDDD	Germany	Deo Spray	19-MAR-05
9	113	BBBB	Australia	Deo Spray	19-MAR-05
10	116	EEEE	Dubai	Deo Spray	19-MAR-05
11	113	BBBB	Australia	Hair Oil	05-NOV-12
12	116	EEEE	Dubai	Hair Oil	05-NOV-12



4. Natural Join

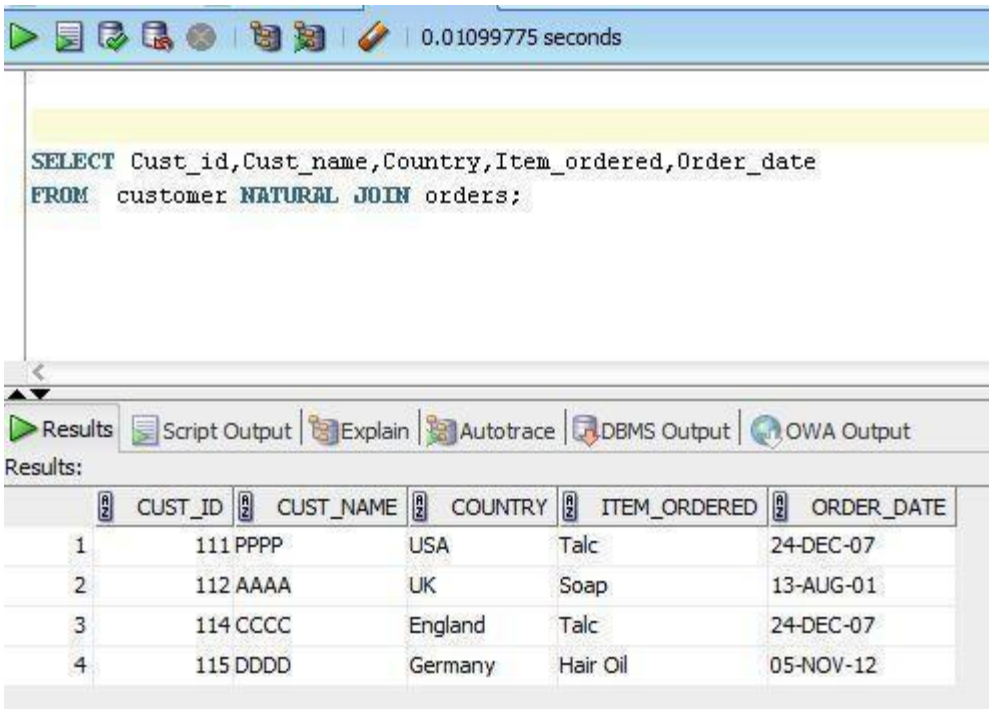
A natural join is just like an equi-join since it compares the common columns of both tables.

Example

SELECT Cust_id, Cust_name, Country, Item_ordered,

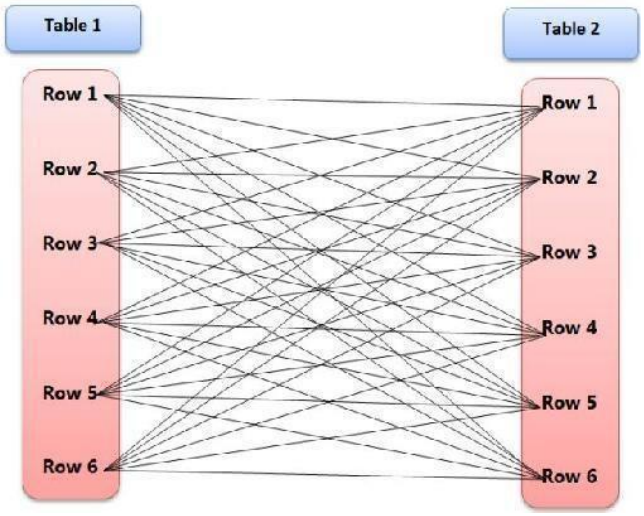
Order_date FROM Customer, NATURAL JOIN Orders;

Execution of the query with result:



5. Cross Join

This join is a little bit different from the other joins since it generates the Cartesian product of two tables as in the following:



Syntax

SELECT * FROM table_name1 CROSS JOIN table_name2;

Example

SELECT Cust_id, Cust_name, Country, Item_ordered, Order_date FROM Customer,
CROSS JOIN Orders;

SQL - Using Joins

The SQL **Joins** clause is used to combine records from two or more tables in a database. A JOIN is a means for combining fields from two tables by using values common to each. Consider the following two tables −

Table 1 − CUSTOMERS Table

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 − ORDERS Table

OID	DATE		CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00		3	3000
100	2009-10-08 00:00:00		3	1500
101	2009-11-20 00:00:00		2	1560
103	2008-05-20 00:00:00		4	2060

Now, let us join these two tables in our SELECT statement as shown below.

SQL> SELECT ID, NAME, AGE, AMOUNT FROM CUSTOMERS, ORDERS WHERE CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

This would produce the following result.

ID	NAME	AGE	AMOUNT
3	kaushik	23	3000
3	kaushik	23	1500
2	Khilan	25	1560
4	Chaitali	25	2060

Here, it is noticeable that the join is performed in the WHERE clause. Several operators can be used to join tables, such as =, <, >, <>, <=, >=, !=, BETWEEN, LIKE, and NOT; they can all be used to join tables. However, the most common operator is the equal to symbol.

There are different types of joins available in SQL −

- INNER JOIN − returns rows when there is a match in both tables.

- LEFT JOIN − returns all rows from the left table, even if there are no matches in the right table.
- RIGHT JOIN − returns all rows from the right table, even if there are no matches in the left table.
- FULL JOIN − returns rows when there is a match in one of the tables.
- SELF JOIN − is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.
- CARTESIAN JOIN − returns the Cartesian product of the sets of records from the two or more joined tables.

SQL - INNER JOINS

The most important and frequently used of the joins is the **INNER JOIN**. They are also referred to as an **EQUIJOIN**.

The INNER JOIN creates a new result table by combining column values of two tables (table1 and table2) based upon the join-predicate. The query compares each row of table1 with each row of table2 to find all pairs of rows which satisfy the join-predicate. When the join-predicate is satisfied, column values for each matched pair of rows of A and B are combined into a result row.

Syntax

SELECT table1.column1, table2.column2... FROM
table1 INNER JOIN table2
ON table1.common_field = table2.common_field;

Example Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	hmedabad	2000.00

2	Khilan	25	Delhi	500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 − ORDERS Table is as follows.

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using the INNER JOIN as follows −

Sql> select id, name, amount, date from customers inner join orders on customers.id = orders.customer_id;

This would produce the following result.

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00

SQL - LEFT JOINS

The SQL **LEFT JOIN** returns all rows from the left table, even if there are no matches in the right table. This means that if the ON clause matches 0 (zero) records in the right table; the join will still

return a row in the result, but with NULL in each column from the right table.

This means that a left join returns all the values from the left table, plus matched values from the right table or NULL in case of no matching join predicate.

Syntax

The basic syntax of a **LEFT JOIN** is as follows.

SELECT table1.column1, table2.column2... FROM
table1 LEFT JOIN table2
ON table1.common_field = table2.common_field;

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables,

Table 1 − CUSTOMERS Table is as follows.ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 − Orders Table is as follows.

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using the LEFT JOIN as follows.

sql> select id, name, amount, date from customers left join orders on customers.id = orders.customer_id;

This would produce the following result –

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL

SQL - RIGHT JOINS

The SQL **RIGHT JOIN** returns all rows from the right table, even if there are no matches in the left table. This means that if the ON clause matches 0 (zero) records in the left table; the join will still return a row in the result, but with NULL in each column from the left table.

This means that a right join returns all the values from the right table, plus matched values from the left table or NULL in case of no matching join predicate.

Syntax

SELECT table1.column1, table2.column2... FROM

table1 RIGHT JOIN table2

```
ON table1.common_field = table2.common_field;
```

Example: Consider the following two tables,

Table 1 − CUSTOMERS Table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2 − ORDERS Table is as follows.

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using the RIGHT JOIN as follows.

```
SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS
RIGHT JOIN ORDERS
ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result −

ID	NAME	AMOUNT	DATE
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00 |
2	Khilan	1560	2009-11-20 00:00:00 |
4	Chaitali	2060	2008-05-20 00:00:00 |

SQL - FULL JOINS

The SQL **FULL JOIN** combines the results of both left and right outer joins.
The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side.

Syntax

SELECT table1.column1, table2.column2... FROM
table1 FULL JOIN table2

ON table1.common_field = table2.common_field;

Here, the given condition could be any given expression based on your requirement.

Example

Consider the following two tables.

Table 1 − CUSTOMERS Table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

7	Muffy	24	Indore	10000.00

Table 2 − ORDERS Table is as follows.

OID	DATE	AMOUNT	CUSTOMER_ID
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using FULL JOIN as follows.

SQL> SELECT ID, NAME, AMOUNT, DATE FROM CUSTOMERS
FULL JOIN ORDERS ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;

This would produce the following result −

ID	NAME	AMOUNT	DATE
1	Ramesh	NULL	NULL
2	Khilan	1560	2009-11-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	NULL	NULL
6	Komal	NULL	NULL
7	Muffy	NULL	NULL
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00

2	Khilan	1560	2009-11-20 00:00:00

4	Chaitali	2060	2008-05-20 00:00:00

If your Database does not support FULL JOIN (MySQL does not support FULL JOIN), then you can use **UNION ALL** clause to combine these two JOINS as shown below.

Sql> select id, name, amount, date from customers left join orders on customers.id = orders.customer_id union all select id, name, amount, date from customers right join orders on customers.id = orders.customer_id

SQL - SELF JOINS

The SQL **SELF JOIN** is used to join a table to itself as if the table were two tables; temporarily renaming at least one table in the SQL statement.

Syntax

SELECT a.column_name, b.column_name... FROM table1 a, table1 b

WHERE a.common_field = b.common_field;

Here, the WHERE clause could be any given expression based on your requirement.

Example Consider the following table.

CUSTOMERS Table is as follows.

ID	NAME	AGE	ADDRESS	ALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Now, let us join this table using SELF JOIN as follows −

sql> select a.id, b.name, a.salary from customers a, customers b where a.salary < b.salary;

This would produce the following result −

ID	NAME	SALARY
2	Ramesh	1500.00
2	kaushik	1500.00
1	Chaitali	2000.00
2	Chaitali	1500.00
3	Chaitali	2000.00
6	Chaitali	4500.00
1	Hardik	2000.00
2	Hardik	1500.00
3	Hardik	2000.00
4	Hardik	6500.00
6	Hardik	4500.00
1	Komal	2000.00
2	Komal	1500.00
3	Komal	2000.00
1	Muffy	2000.00
2	Muffy	1500.00
3	Muffy	2000.00
4	Muffy	6500.00
5	Muffy	8500.00
6	Muffy	4500.00

SQL - CARTESIAN or CROSS JOINS

The CARTESIAN JOIN or CROSS JOIN returns the Cartesian product of the sets of records from two or more joined tables. Thus, it equates to an inner join where the join-condition always evaluates to either True or where the join-condition is absent from the statement.

Syntax

The basic syntax of the **CARTESIAN JOIN** or the **CROSS JOIN** is as follows −

SELECT table1.column1, table2.column2...FROM table1, table2 [, table3]

Example

Consider the following two tables.

Table 1 − CUSTOMERS table is as follows.

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Table 2: ORDERS Table is as follows –

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

Now, let us join these two tables using CARTESIAN JOIN as follows −

sql> select id, name, amount, date from customers, orders;

ID	NAME	AMOUNT	DATE
1	Ramesh	3000	2009-10-08 00:00:00
1	Ramesh	1500	2009-10-08 00:00:00
1	Ramesh	1560	2009-11-20 00:00:00
1	Ramesh	2060	2008-05-20 00:00:00
2	Khilan	3000	2009-10-08 00:00:00
2	Khilan	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
2	Khilan	2060	2008-05-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
3	kaushik	1560	2009-11-20 00:00:00
3	kaushik	2060	2008-05-20 00:00:00
4	Chaitali	3000	2009-10-08 00:00:00
4	Chaitali	1500	2009-10-08 00:00:00
4	Chaitali	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	3000	2009-10-08 00:00:00
5	Hardik	1500	2009-10-08 00:00:00
5	Hardik	1560	2009-11-20 00:00:00
5	Hardik	2060	2008-05-20 00:00:00
6	Komal	3000	2009-10-0800:00:00
6	Komal	1500	2009-10-0800:00:00
6	Komal	1560	2009-11-2000:00:00
7	Muffy	2060	2008-05-2000:00:00

EMBEDDED SQL

The first technique for sending SQL statements to the DBMS is embedded SQL. The SQL standard defines embeddings of SQL in a variety of programming languages such as C,Java, and Cobol.

A language to which SQL queries are embedded is referred to as a host language, and the SQL structures permitted in the host language comprise embedded SQL.

The following techniques are used to embed SQL statements in a host language:

- Embedded SQL statements are processed by a special SQL precompiler. All SQL statements begin with an introducer and end with a terminator, both of which flag the SQL statement for the precompiler. For example, the introducer is "EXEC SQL" in C and "& and the terminator is a semicolon (;) in C.
- Variables from the application program, called host variables, can be used in embedded SQL statements wherever constants are allowed.
- Queries that return a single row of data are handled with a singleton SELECT statement; this statement specifies both the query and the host variables in which to return data.
- Queries that return multiple rows of data are handled with cursors. A cursor keeps track of the current row within a result set. The DECLARE CURSOR statement defines the query, the OPEN statement begins the query processing, the FETCH statement retrieves successive rows of data, and the CLOSE statement ends query processing.
- While a cursor is open, positioned update and positioned delete statements can be used to update or delete the row currently selected by the cursor.

Embedded SQL Example

EXEC SQL statement is used to identify embedded SQL request to the preprocessor EXEC

SQL <embedded SQL statement > END_EXEC

Note: this varies by language (for example, the Java embedding uses # SQL { …. };)

From within a host language, find the names and cities of customers with more than the variable amount dollars in some account.

Specify the query in SQL and declare a *cursor* for it EXEC

SQL

declare *c* **cursor for** **select** *depositor.customer_name, customer_city* **from** *depositor, customer, account* **where** *depositor.customer_name = customer.customer_name* **and** *depositor account_number = account.account_number* **and** *account.balance > :amount*

END_EXEC

The **open** statement causes the query to be evaluated EXEC

SQL **open** *c* END_EXEC

The **fetch** statement causes the values of one tuple in the query result to be placed on host language variables.

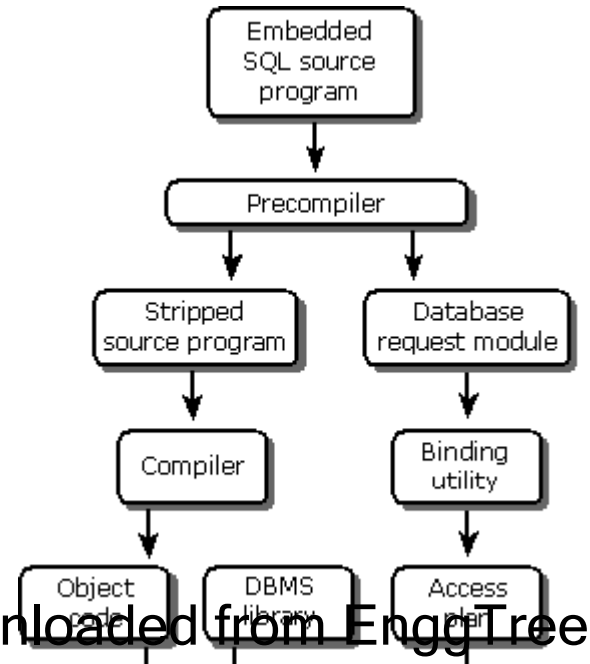
EXEC SQL **fetch** *c* **into** :*cn, :cc* END_EXEC Repeated calls to **fetch** get successive tuples in the query result

A variable called SQLSTATE in the SQL communication area (SQLCA) gets set to '02000' to indicate no more data is available

The **close** statement causes the database system to delete the temporary relation that holds the result of the query.

EXEC SQL **close** *c* END_EXEC

Compiling an Embedded SQL Program



DYNAMIC SQL

Dynamic SQL is the process that we follow for programming SQL queries in such a way that the queries are built dynamically with the application operations.

It helps us to manage big industrial applications and manage the transactions without any added overhead.

With dynamic SQL we are free to create flexible SQL queries and the names of the variables or any other parameters are passed when the application runs. Allows programs to construct and submit SQL queries at run time. We can use stored procedures to create dynamic queries which can run when we desire.

For Dynamic SQL, we use the exec keyword.

When we use static SQL it is not altered from one execution to others, but in the case of dynamic SQL, we can alter the query in each execution.

Why do we need Dynamic SQL?

We need to use Dynamic SQL for the following use cases:

When we need to run dynamic queries on our database, mainly DML queries.

When we need to access an object which is not in existence during the compile time.

Whenever we need to optimize the run time of our queries.

When we need to instantiate the created logic blocks.

When we need to perform operations on application fed data using invoker rights.

Example of the use of dynamic SQL from within a C program.

char * sqlprog = “update account set balance = balance * 1.05 where account_number = ?” EXEC

SQL prepare dynprog from :sqlprog;char account [10] = “A-101”;

EXEC SQL execute dynprog using :account;

The dynamic SQL program contains a ?, which is a place holder for a value that is provided

when the SQL program is executed.

Dynamic SQL statements can be built at run time and placed in a string host variable. They are sent to the DBMS for processing. Because the DBMS must generate an access plan at run time for dynamic SQL statements, dynamic SQL is generally slower than static SQL.

The simplest way to execute a dynamic SQL statement is with an EXECUTE IMMEDIATE statement. This statement passes the SQL statement to the DBMS for compilation and execution. One disadvantage of the EXECUTE IMMEDIATE statement is that the DBMS must go through each of the five steps of processing an SQL statement each time the statement is executed.

To address this situation, dynamic SQL offers an optimized form of execution called prepared execution, which uses the following steps:

- The program constructs an SQL statement in a buffer, just as it does for the EXECUTE IMMEDIATE statement. Instead of host variables, a question mark (?) can be substituted for a constant anywhere in the statement text to indicate that a value for the constant will be supplied later. The question mark is called as a parameter marker.
- The program passes the SQL statement to the DBMS with a PREPARE statement, which requests that the DBMS parse, validate, and optimize the statement and generate an execution plan for it. The program then uses an EXECUTE statement (not an EXECUTE IMMEDIATE statement) to execute the PREPARE statement at a later time. It passes parameter values for the statement through a special data structure called the SQL Data Area or SQLDA.
- The program can use the EXECUTE statement repeatedly, supplying different parameter values each time the dynamic statement is executed.
- Prepared execution is still not the same as static SQL. In static SQL, the first four steps of processing an SQL statement take place at compile time. In prepared execution, these steps still take place at run time, but they are performed only once; execution of the plan takes place only when EXECUTE is called. This helps eliminate some of the performance disadvantages inherent in the architecture of dynamic SQL.

Difference between Static SQL and Dynamic SQL

Sr. No.	Key	Static SQL	Dynamic SQL
1	Database Access	In Static SQL, database access procedure is predetermined in the statement.	In Dynamic SQL, how a database will be accessed, can be determine only at run time.

Sr. No.	Key	Static SQL	Dynamic SQL
2	Efficiency	Static SQL statements are more faster and efficient.	Dynamic SQL statements are less efficient.
3	Compilation	Static SQL statements are compiled at compile time.	Dynamic SQL statements are compiled at run time.
4	Application Plan	Application Plan parsing, validation, optimization and generation are compile time activities.	Application Plan parsing, validation, optimization and generation are run time activities.
5	Use Cases	Static SQL is used in case of uniformly distributed data.	Dynamic SQL is used in case of non-uniformly distributed data.
6	Dynamic Statements	Statements like EXECUTE IMMEDIATE, EXECUTE, PREPARE are not used.	Statements like EXECUTE IMMEDIATE, EXECUTE, PREPARE are used

PANIMALAR INSTITUTE OF TECHNOLOGY

DEPARTMENT OF INFORMATION TECHNOLOGY

CS3492 DATABASE MANAGEMENT SYSTEMS

LECTURE NOTES

UNIT II DATABASE DESIGN

Entity-Relationship model – E-R Diagrams – Enhanced-ER Model – ER-to-Relational Mapping – Functional Dependencies – Non-loss Decomposition – First, Second, Third Normal Forms, Dependency Preservation – Boyce/Codd Normal Form – Multi-valued Dependencies and Fourth Normal Form – Join Dependencies and Fifth Normal Form

ENTITY-RELATIONSHIP MODEL

The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them.

Entity

An entity can be a real-world object that can be easily identifiable. For example, 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.

Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

Types of Attributes

Simple attribute − Simple attributes are atomic values, which cannot be divided further.

For example, a student's phone number is an atomic value of 10 digits.

Composite attribute − Composite attributes are made of more than one simple

attribute. For example, 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. For

example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.

Single-value attribute − Single-value attributes contain single value. For example − Social_Security_Number.

Multi-value attribute − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

Entity-Set and Keys

Entity-Set :

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Keys :

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set. For example, the roll_number of a student makes him/her identifiable among students.

Super Key − A set of attributes (one or more) that collectively identifies an entity in an entity set.

Candidate Key − A minimal super key is called a candidate key. An entity set may have more than one candidate key.

Primary Key − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Relationship

The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

Relationship Set

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Degree of Relationship

The number of participating entities in a relationship defines the degree of the relationship.

Binary = degree 2

Ternary = degree 3

n-ary = degree

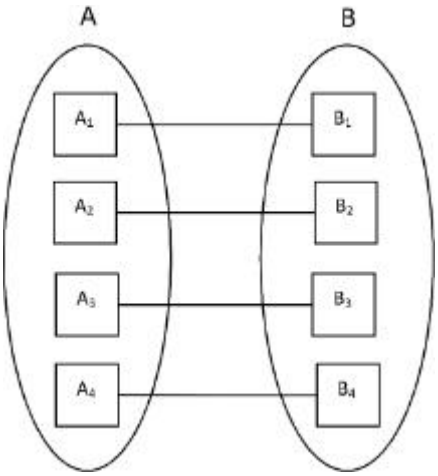
Mapping Cardinality (cardinality constraint)

It represents the number of entities of another entity set which are connected to an entity using a relationship set.

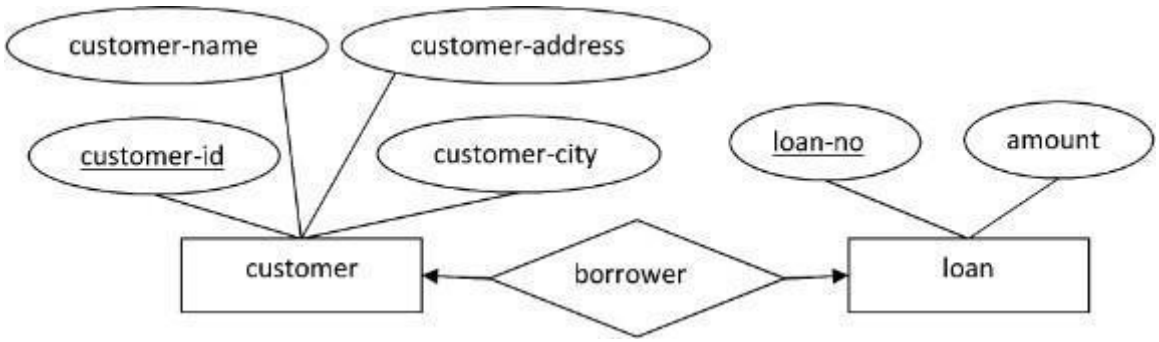
For a binary relationship set the mapping cardinality must be one of the following types:

1. One to one
2. One to many
3. Many to one
4. Many to many

1. ***One-to-one relationship***

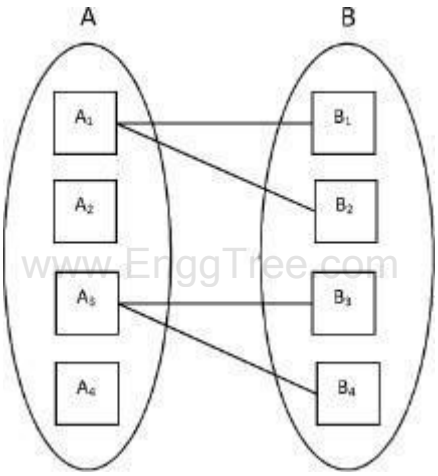


An entity in A is associated with at most (only) one entity in B and an entity in B is associated with at most (only) one entity in A.

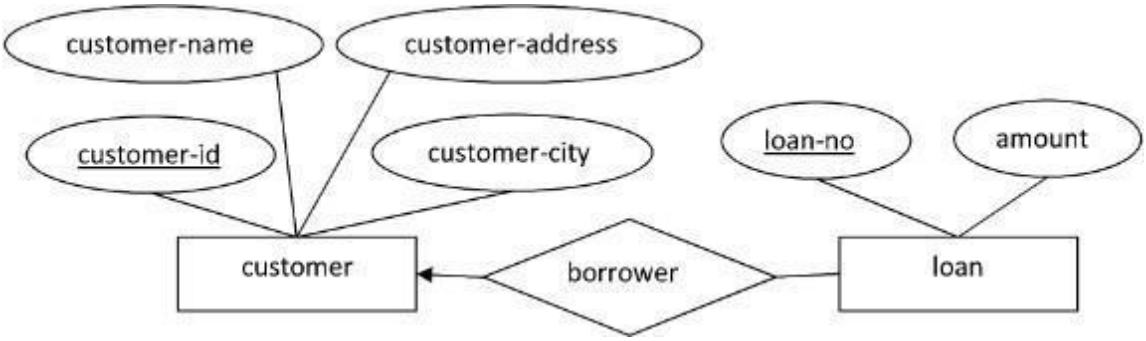


A customer is connected with only one loan using the relationship borrower and a loan is connected with only one customer using borrower.

2. ***One-to-many relationship***

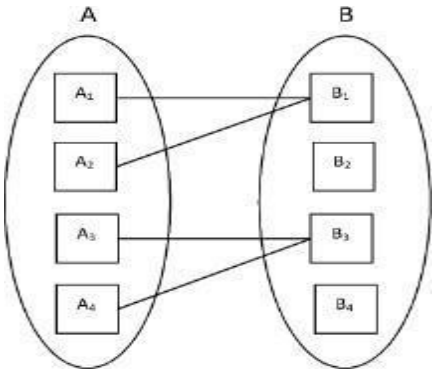


An entity in A is associated with any number (zero or more) of entities in Band an entity in Bis associated with at most one (only) entity in A.

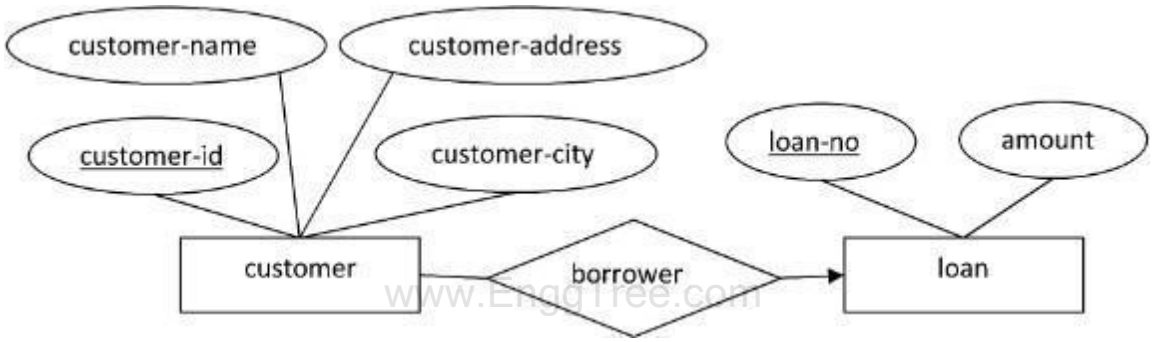


In the one-to-many relationship a loan is connected with only one customer using borrower and a customer is connected with more than one loans using borrower.

3. *Many-to-one relationship*

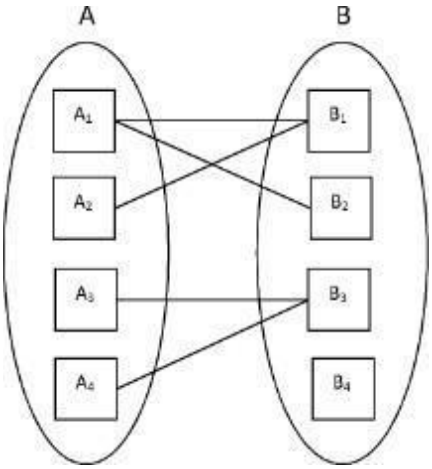


An entity in A is associated with at most (only) one entity in B and an entity in B is associated with any number (zero or more) of entities in A.

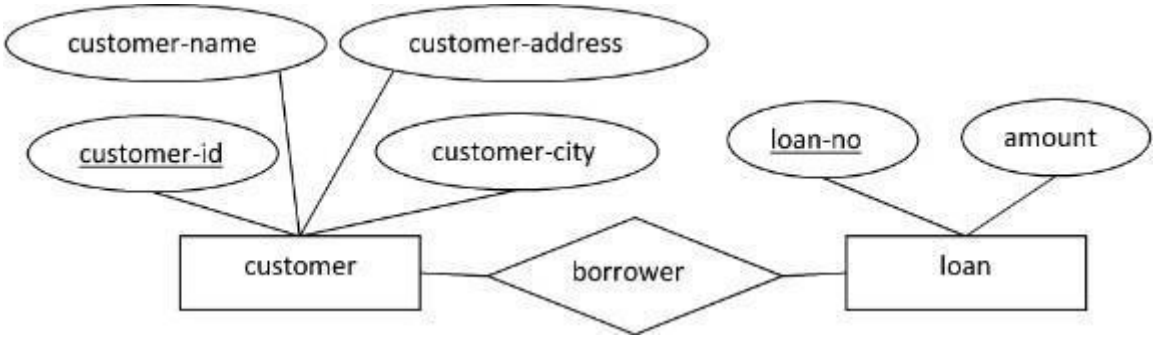


In a many-to-one relationship a loan is connected with more than one customer using borrower and a customer is connected with only one loan using borrower.

4. *Many-to-many relationship*



An entity in A is associated with any number (zero or more) of entities in Band an entity in Bis associated with any number (zero or more) of entities in A.



A customer is connected with more than one loan using borrower and a loan is connected with more than one customer using borrower.

E-R Diagrams

E-R diagram is the short form of "Entity-Relationship" diagram. An E-R diagram efficiently shows the relationships between various entities stored in a database.

E-R diagrams are used to model real-world objects like a person, a car, a company etc. and the relation between these real-world objects. An e-r diagram has following features:

E-R diagrams are used to represent E-R model in a database, which makes them easy to be converted into relations (tables).

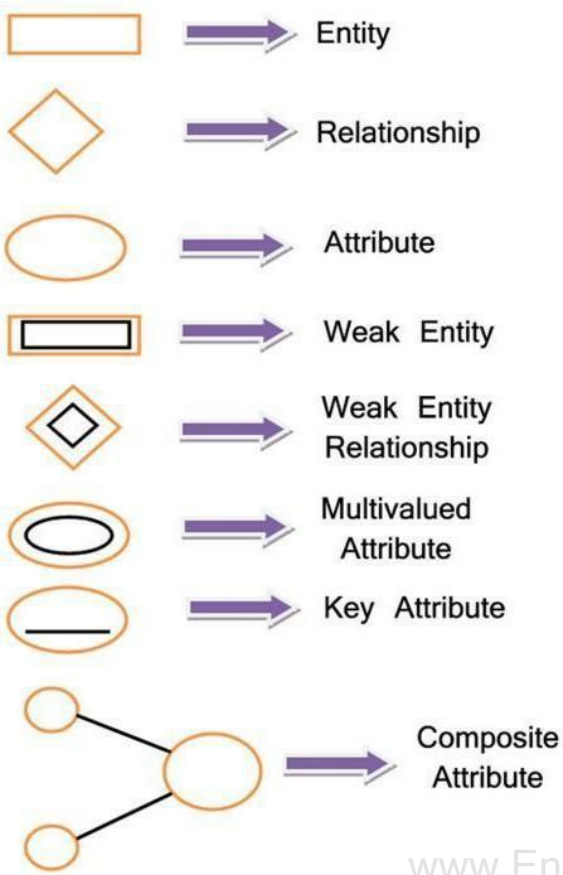
E-R diagrams provide the purpose of real-world modeling of objects which makes them intently useful.

E-R diagrams require no technical knowledge & no hardware support.

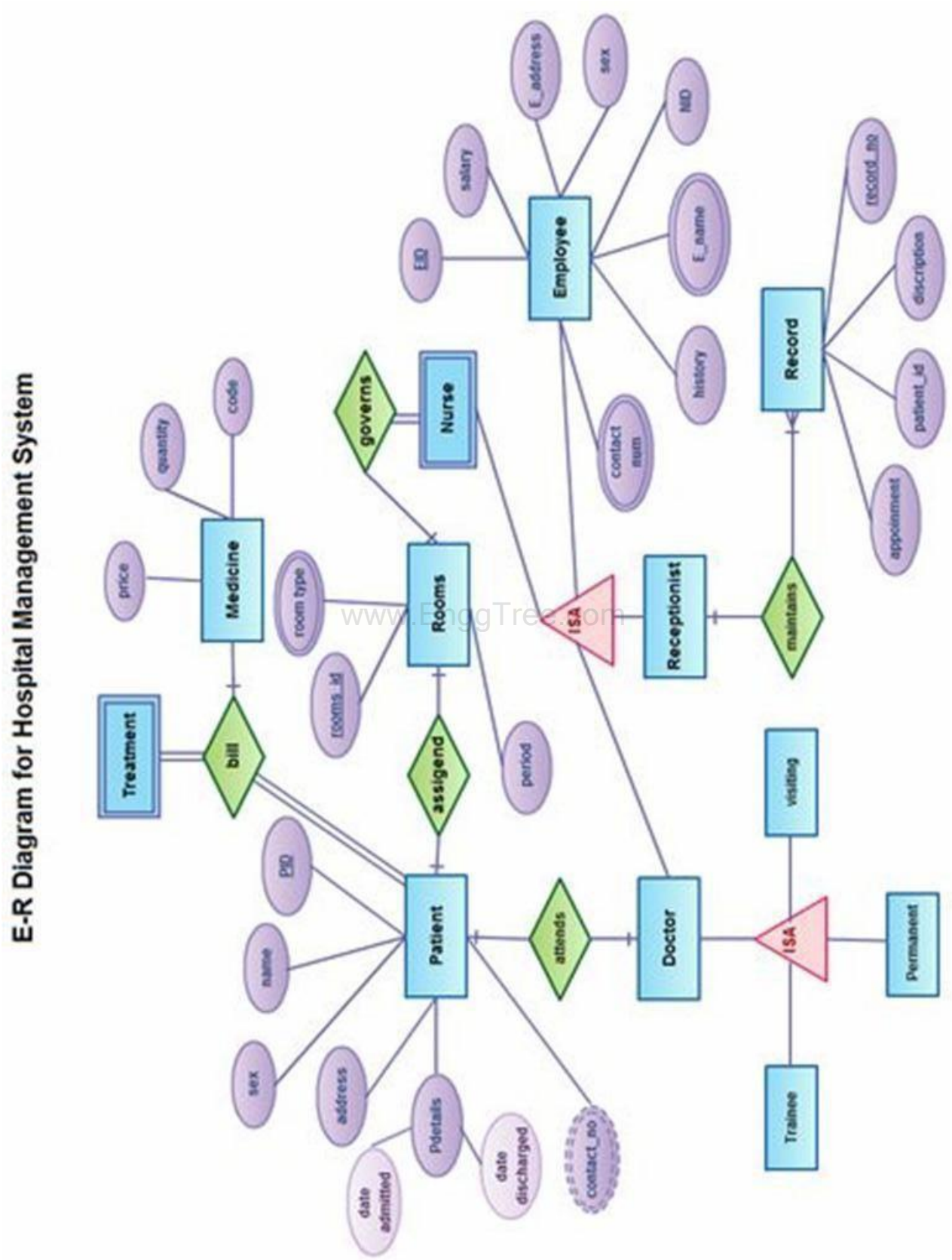
These diagrams are very easy to understand and easy to create even by a naive user.

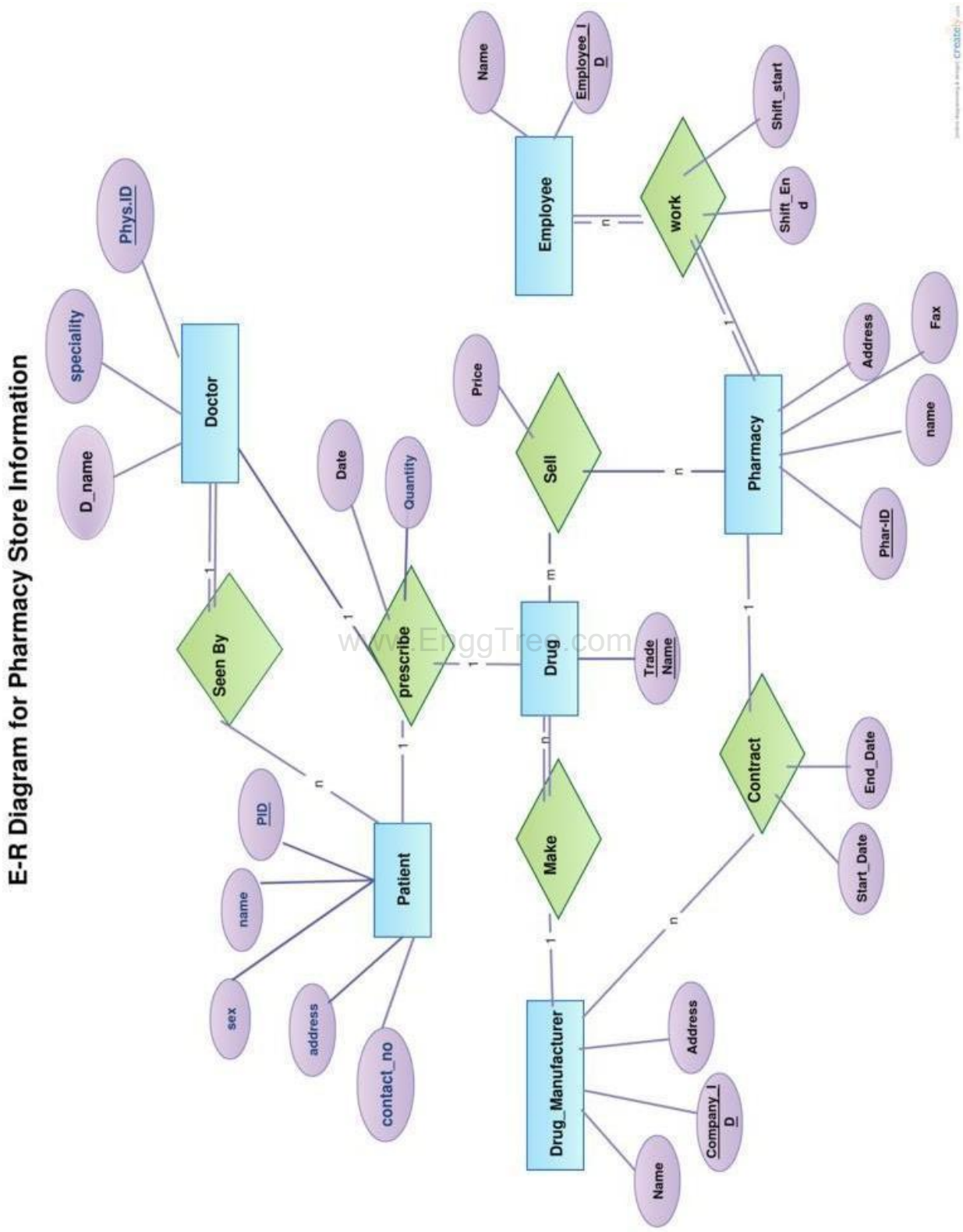
It gives a standard solution of visualizing the data logically.

E R Diagrams Symbols, And Notations



E R Diagrams Example:





ENHANCED ENTITY RELATIONSHIP MODEL (EER MODEL)

EER is a high-level data model that incorporates the extensions to the original ER model.

It is a diagrammatic technique for displaying the following concepts

Sub Class and Super Class

Specialization and Generalization

Union or Category

Aggregation

These concepts are used when the comes in EER schema and the resulting schema diagrams called as EER Diagrams.

Features of EER Model

EER creates a design more accurate to database schemas.

It reflects the data properties and constraints more precisely.

It includes all modeling concepts of the ER model.

Diagrammatic technique helps for displaying the EER schema.

It includes the concept of specialization and generalization.

It is used to represent a collection of objects that is union of objects of different of different entity types.

A. Sub Class and Super Class

Sub class and Super class relationship leads the concept of Inheritance.

The relationship between sub class and super class is denoted with  symbol.

1. Super Class

- Super class is an entity type that has a relationship with one or more subtypes.
- An entity cannot exist in database merely by being member of any super class.

For example: Shape super class is having sub groups as Square, Circle, and Triangle.

2. Sub Class

- Sub class is a group of entities with unique attributes.

- Sub class inherits properties and attributes from its super class.

For example: Square, Circle, Triangle are the sub class of Shape super class.

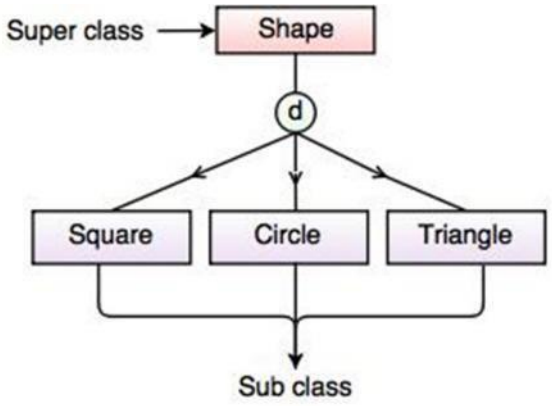


Fig. Super class/Sub class Relationship

B. Specialization and Generalization

1. Generalization

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
- It is a bottom approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.
- It minimizes the difference between the entities by identifying the common features.

For example:

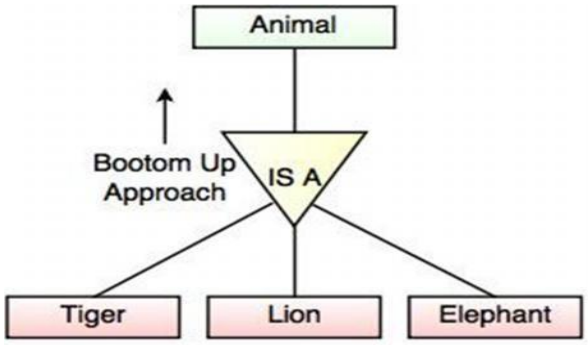


Fig. Generalization

In the above example, Tiger, Lion, Elephant can all be generalized as Animals.

2. Specialization

Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic.

It is a top down approach, in which one higher entity can be broken down into two lower level entity.

It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.

It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

For example

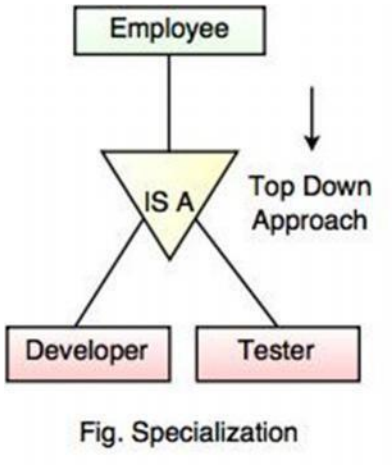


Fig. Specialization

In the above example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

C. Category or Union

Category represents a single super class or sub class relationship with more than one super class.

It can be a total or partial participation.

For example Car booking, Car owner can be a person, a bank (holds a possession on a Car) or a company. Category (sub class) → Owner is a subset of the union of the three super classes → Company, Bank, and Person. A Category member must exist in at least one of its super classes.

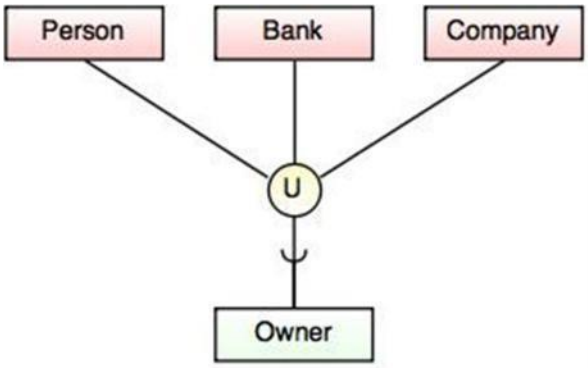


Fig. Categories (Union Type)

D. Aggregation

Aggregation is a process that represent a relationship between a whole object and its component parts.

It abstracts a relationship between objects and viewing the relationship as an object.

It is a process when two entity is treated as a single entity.

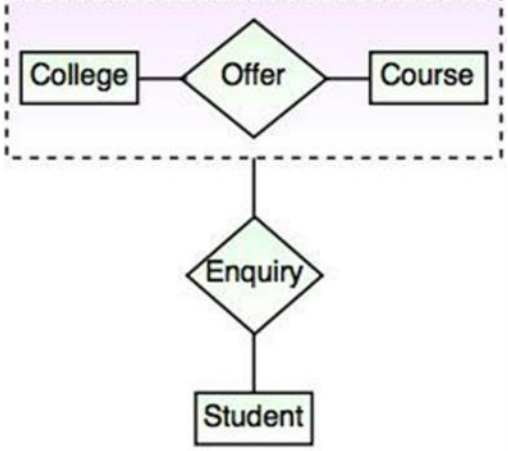


Fig. Aggregation

In the above example, the relation between College and Course is acting as an Entity in Relation with Student.

ER-to-Relational Mapping

The ER Model is intended as a description of real-world entities. Although it is constructed in such a

way as to allow easy translation to the relational schema model, this is not an entirely

trivial process. The ER diagram represents the conceptual level of database design meanwhile the relational schema is the logical level for the database design.

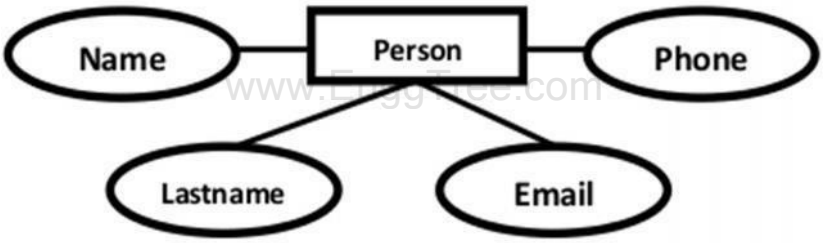
1. Entities and Simple Attributes:

An entity type within ER diagram is turned into a table. You may preferably keep the same name for the entity or give it a sensible name but avoid DBMS reserved words as well as avoid the use of special characters.

Each attribute turns into a column (attribute) in the table. The key attribute of the entity is the primary key of the table which is usually underlined. It can be composite if required but can never be null.

It is highly recommended that every table should start with its primary key attribute conventionally named as TablenameID.

Taking the following simple ER diagram:



The initial relational schema is expressed in the following format writing the table names with the attributes list inside a parentheses as shown below for

Persons(<u>personid</u> , name, lastname, email)

Persons and Phones are Tables. name, lastname, are Table Columns (Attributes).personid is the primary key for the table : Person

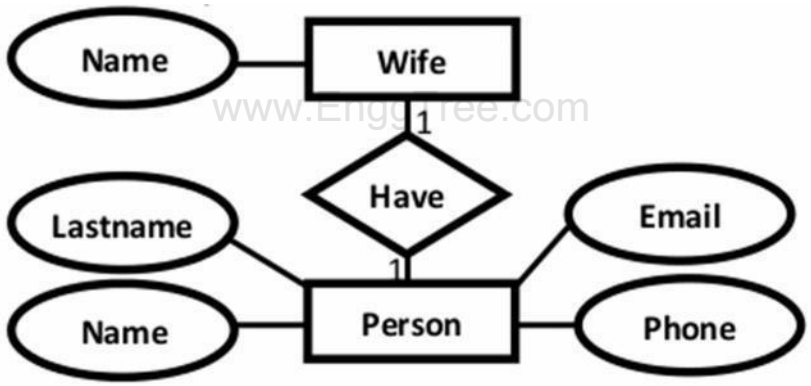
2. Multi-Valued Attributes

A multi-valued attribute is usually represented with a double-line oval.

If you have a multi-valued attribute, take the attribute and turn it into a new entity or table of its own. Then make a 1:N relationship between the new entity and the existing one. In simple words.
1. Create a table for the attribute. 2. Add the primary (id) column of the parent entity as a foreign key within the new table as shown below:

Persons(<u>personid</u> , name, lastname, email) Phones (<u>phoneid</u> , *personid*, phone)
personid within the table Phones is a foreign key referring to the personid of Persons

3. 1:1 Relationships



To keep it simple and even for better performances at data retrieval, I would personally recommend using attributes to represent such relationship. For instance, let us consider the case where the Person has or optionally has one wife. You can place the primary key of the wife within the table of the Persons which we call in this case Foreign key as shown below.

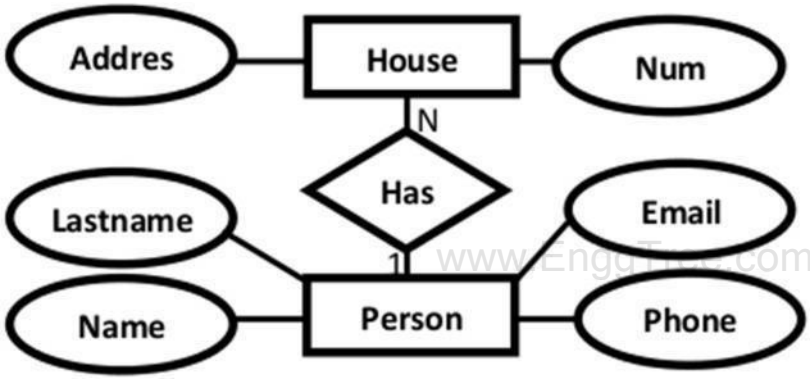
Persons(<u>personid</u> , name, lastname, email , *wifeid*) Wife (<u>wifeid</u> , name)
Or vice versa to put the **personid** as a foreign key within the Wife table as shown below:
Persons(<u>personid</u> , name, lastname, email)

Wife (wifeid , name , *personid*)

For cases when the Person is not married i.e. has no wifeID, the attribute can set to NULL

4. 1:N Relationships

This is the tricky part ! For simplicity, use attributes in the same way as 1:1 relationship but we have only one choice as opposed to two choices. For instance, the Person can have a **House** from zero to many , but a **House** can have only one **Person**. To represent such relationship the **personid**as the Parent node must be placed within the Child table as a foreign key but **not the other way around as shown next:**



It should convert to :

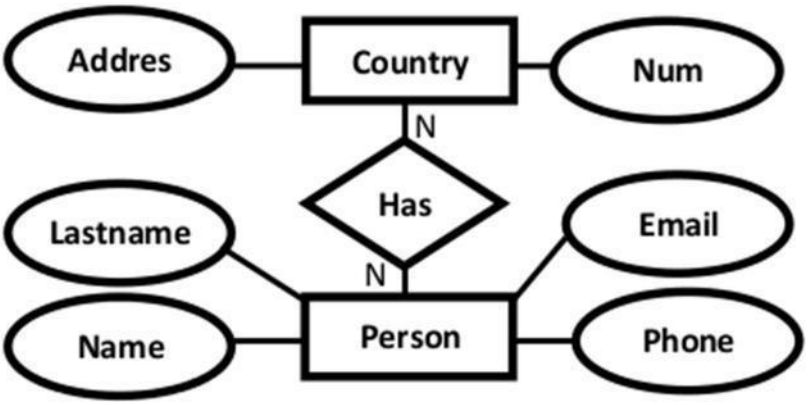
Persons(personid , name, lastname, email

) House (houseid , num , address,

***personid*)**

5. N:N Relationships

We normally use tables to express such type of relationship. This is the same for N − ary relationship of ER diagrams. For instance, The Person can live or work in many countries. Also, a country can have many people. To express this relationship within a relational schema we use a separate table as shown below:



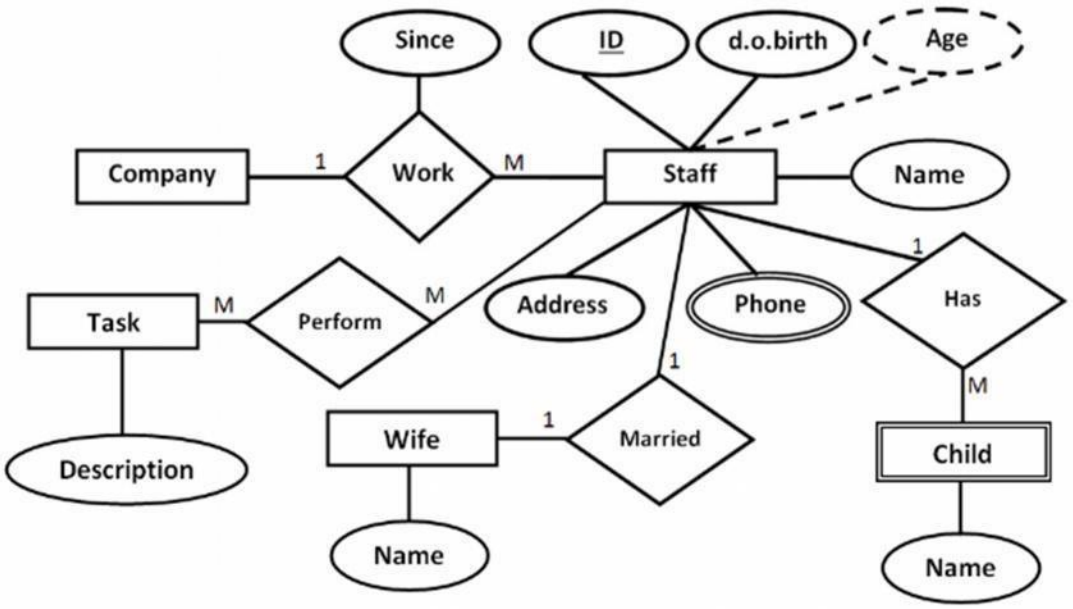
It should convert into :

Persons(personid , name, lastname, email) Countries (countryid , name, code) HasRelat (hasrelatid , **personid , countryid**)

Relationship with attributes:

It is recommended to use table to represent them to keep the design tidy and clean **regardless of the cardinality** of the relationship.

Case Study



The relational schema for the ER Diagram is given below as:

Company(CompanyID , name , address) Staff(StaffID , dob , address , ***WifeID***) Child(ChildID , name , ***StaffID***) Wife

(WifeID , name)

Phone(PhoneID , phoneNumber , ***StaffID***) Task (TaskID , description)

Work(WorkID , ***CompanyID , StaffID*** , since)

Perform(PerformID , ***StaffID , TaskID***)

Functional Dependency

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table.

X → Y

The left side of FD is known as a determinant, the right side of the production is known as a dependent.

For example:

Assume we have an employee table with attributes: Emp_Id, Emp_Name, Emp_Address.

Here Emp_Id attribute can uniquely identify the Emp_Name attribute of employee table because if we know the Emp_Id, we can tell that employee name associated with it.

Functional dependency can be written as:

Emp_Id → Emp_Name

We can say that Emp_Name is functionally dependent on Emp_Id. Types of Functional dependency

1. Trivial functional dependency
2. Non-trivial functional dependency

1. Trivial functional dependency

- A → B has trivial functional dependency if B is a subset of A.
- The following dependencies are also trivial like: A → A, B → B

Example:

1. Consider a table with two columns Employee_Id and Employee_Name.
2. {Employee_id, Employee_Name} → Employee_Id is a trivial functional dependency as Employee_Id is a subset of {Employee_Id, Employee_Name}.
3. Also, Employee_Id → Employee_Id and Employee_Name → Employee_Name are trivia l dependencies too.

2. Non-trivial functional dependency

- A → B has a non-trivial functional dependency if B is not a subset of A.
- When A intersection B is NULL, then A → B is called as complete non-trivial.

Example:

1. ID → Name,
2. Name → DOB

Armstrong'sAxioms

If F is a set of functional dependencies then the closure of F, denoted as F^+, is the set of all functional dependencies logically implied by F. Armstrong's Axioms are a set of rules, that when applied repeatedly, generates a closure of functional dependencies.

- **Reflexive rule** − If alpha is a set of attributes and beta is_subset_of alpha, then alpha holds beta.
- **Augmentation rule** − If a → b holds and y is attribute set, then ay → by also holds. That is adding attributes in dependencies, does not change the basic dependencies.
- **Transitivity rule** − Same as transitive rule in algebra, if a → b holds and b → c holds, then a → c also holds. a → b is called as a functionally that determines b.

Non-loss Decomposition

Decomposition in DBMS removes redundancy, anomalies and inconsistencies from a database by dividing the table into multiple tables.

The following are the types:

Lossless Decomposition

Decomposition is lossless if it is feasible to reconstruct relation R from decomposed tables using Joins. This is the preferred choice. The information will not lose from the relation when decomposed. The join would result in the same original relation.

Let us see an example:

<EmpInfo>

Emp_ID	Emp_Name	Emp_Age	Emp_Location	Dept_ID	Dept_Name
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR

E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

<EmpDetails>

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	Emp_ID	Dept_Name
Dpt1	E001	Operations
Dpt2	E002	HR
Dpt3	E003	Finance

Now, Natural Join is applied on the above two tables:

The result will be:

Emp_I D	Emp_Nam e	Emp_Ag e	Emp_Locatio n	Dept_I D	Dept_Nam e
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Therefore, the above relation had lossless decomposition i.e. no loss of information.

Lossy Decomposition

As the name suggests, when a relation is decomposed into two or more relational schemas, the loss of information is unavoidable when the original relation is retrieved.

Let us see an example:

<EmpInfo>

Emp_I D	Emp_Nam e	Emp_Ag e	Emp_Locatio n	Dept_I D	Dept_Nam e
E001	Jacob	29	Alabama	Dpt1	Operations
E002	Henry	32	Alabama	Dpt2	HR
E003	Tom	22	Texas	Dpt3	Finance

Decompose the above table into two tables:

<EmpDetails>

Emp_ID	Emp_Name	Emp_Age	Emp_Location
E001	Jacob	29	Alabama
E002	Henry	32	Alabama
E003	Tom	22	Texas

<DeptDetails>

Dept_ID	Dept_Name
Dpt1	Operations
Dpt2	HR
Dpt3	Finance

- Now, you won't be able to join the above tables, since **Emp_ID** isn't part of the **DeptDetails** relation.
- Therefore, the above relation has lossy decomposition.

NORMALIZATION

Database Normalization is a technique of organizing the data in the database. Normalization is a systematic approach of decomposing tables to eliminate data redundancy(repetition) and undesirable characteristics like Insertion, Update and Deletion Anamolies. It is a multi-step process that puts data into tabular form, removing duplicated data from the relation tables.

Normalization is used for mainly two purposes,

- Eliminating redundant(useless) data.
- Ensuring data dependencies make sense i.e data is logically stored.

Problems Without Normalization

If a table is not properly normalized and have data redundancy then it will not only eat up extra memory space but will also make it difficult to handle and update the database, without facing data loss. Insertion, Updation and Deletion Anamolies are very frequent if database is not normalized. To understand these anomalies let us take an example of a **Student** table.

rollno	name	branch	hod	office_tel
401	Akon	CSE	Mr. X	53337
402	Bkon	CSE	Mr. X	53337
403	Ckon	CSE	Mr. X	53337
404	Dkon	CSE	Mr. X	53337

In the table above, we have data of 4 Computer Sci. students. As we can see, data for the fields branch, hod(Head of Department) and office_tel is repeated for the students who are in the same branch in the college, this is **Data Redundancy**.

1. Insertion Anomaly

Suppose for a new admission, until and unless a student opts for a branch, data of the student cannot be inserted, or else we will have to set the branch information as **NULL**.
Also, if we have to insert data of 100 students of same branch, then the branch information will be repeated for all those 100 students.
These scenarios are nothing but **Insertion anomalies**.

2. Updation Anomaly

What if Mr. X leaves the college? or is no longer the HOD of computer science department? In that case all the student records will have to be updated, and if by mistake we miss any record, it will lead to data inconsistency. This is Updation anomaly.

.3. Deletion Anomaly

In our **Student** table, two different informations are kept together, Student information and Branch information. Hence, at the end of the academic year, if student records are deleted, we will also lose the branch information. This is Deletion anomaly.

Normalization

- Normalization is the process of organizing the data in the database.
- Normalization is used to minimize the redundancy from a relation or set of relations. It is also used to eliminate the undesirable characteristics like Insertion, Update and Deletion Anomalies.
- Normalization divides the larger table into the smaller table and links them using relationship.

- The normal form is used to reduce redundancy from the database table.

Types of Normal Forms

Normal Form	Description
1NF	A relation is in 1NF if it contains an atomic value.
2NF	A relation will be in 2NF if it is in 1NF and all non-key attributes are fully functional dependent on the primary key.
3NF	A relation will be in 3NF if it is in 2NF and no transition dependency exists.
4NF	A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
5NF	A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.

2. First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all non-key attributes are fully functional dependent on the primary key

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.
To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

A relation is in third normal form if it holds at least one of the following conditions for every non-trivial function dependency X → Y.

1. X is a super key.
2. Y is a prime attribute, i.e., each element of Y is part of some candidate key.

Example:

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:
{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}. so on
Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.
Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.

- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.

Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate key: {EMP-ID, EMP-DEPT}

The table is not in BCNF because neither EMP_DEPT nor EMP_ID alone are keys.

To convert the given table into BCNF, we decompose it into three tables:

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

1. EMP_ID → EMP_COUNTRY
2. EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Candidate keys:

For the first table: EMP_ID

For the second table: EMP_DEPT

For the third table: {EMP_ID, EMP_DEPT}

Now, this is in BCNF because left side part of both the functional dependencies is a key.

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no multi-valued dependency.
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

So to make the above table into 4NF, we can decompose it into two tables:

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.
So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

P1

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen