Fig: An example of the Network model representing a university

1.6 DATABASE SYSTEM ARCHITECTURE OR COMPONENTS OF DBMS

Figure 2.3 illustrates the typical DBMS components. The figure is divided into two parts. The top part of the figure refers to the various users of the database environment and their interfaces. The lower part shows the internals of the DBMS responsible for storage of data and processing of transactions.

Upper Module:

Casual users:

They work with interactive interfaces to formulate queries.

Application programmers:

They create programs using some host programming languages

Parametric users:

They do data entry work by supplying parameters to predefined transactions.

DBA staff:

They work on defining the database and tuning it by making changes to its definition using the DDL and other privileged commands.

DDL Compiler:

The DDL compiler processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog. The catalog includes information such as the names and sizes of files, names and data types of data items, storage details of each file, mapping information among schemas, and constraints. In addition, the catalog stores many other types of information that are needed by the DBMS modules, which can then look up the catalog information as needed.

Query Compiler & Optimizer:

Query compiler compiles the queries of casual users and translates them into an internal

form. This internal query is subjected to query optimization. The **query optimizer** is concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution. It consults the system catalog for statistical and other physical information about the stored data and generates executable code that performs the necessary operations for the query and makes calls on the runtime processor.

Precompiler, DML Compiler & Host Language Compiler:

Application programmers write programs in host languages such as Java, C, or C++ that are submitted to a precompiler. The **precompiler** extracts DML commands from an application program written in a host programming language. These commands are sent to the DML compiler for compilation into object code for database access.

The rest of the program is sent to the host language compiler. The object codes for the DML commands and the rest of the program are linked, forming a canned transaction whose executable code includes calls to the runtime database processor. Canned transactions are executed repeatedly by parametric users, who simply supply the parameters to the transactions. Each execution is considered to be a separate transaction.

Lower Module:

Runtime database processor:

It executes the privileged commands, the executable query plans, and the canned transactions with runtime parameters. It works with the **system catalog** and may update it with statistics. It also works with the **stored data manager**, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory.

The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory. Some DBMSs have their own buffer management module while others depend on the OS for buffer management. We have shown **concurrency control** and **backup and recovery systems** separately as a module in this figure. They are integrated into the working of the runtime database processor for purposes of transaction management.

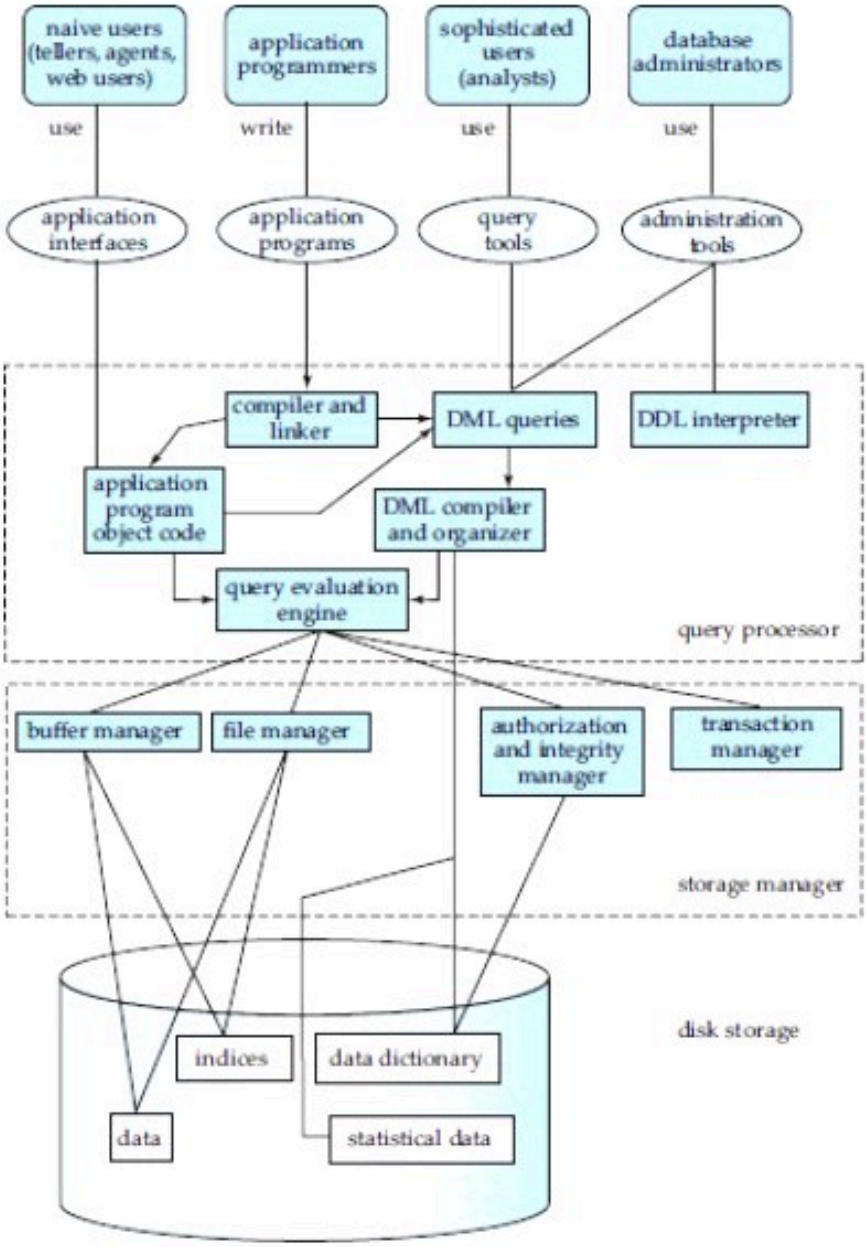


Figure 1.5 System structure.

Data Independence

- Data independence can be explained using the three-schema architecture.
- Data independence refers characteristic of being able to modify the schema at one level of the database system without altering the schema at the next higher level.

There are two types of data independence:

1. Logical Data Independence

- Logical data independence refers characteristic of being able to change the conceptual schema without having to change the external schema.
- Logical data independence is used to separate the external level from the conceptual view.
- If we do any changes in the conceptual view of the data, then the user view of the data would not be affected.
- Logical data independence occurs at the user interface level.

2. Physical Data Independence

- Physical data independence can be defined as the capacity to change the internal schema without having to change the conceptual schema.
- If we do any changes in the storage size of the database system server, then the Conceptual structure of the database will not be affected.
- Physical data independence is used to separate conceptual levels from the internal levels.
- Physical data independence occurs at the logical interface level.

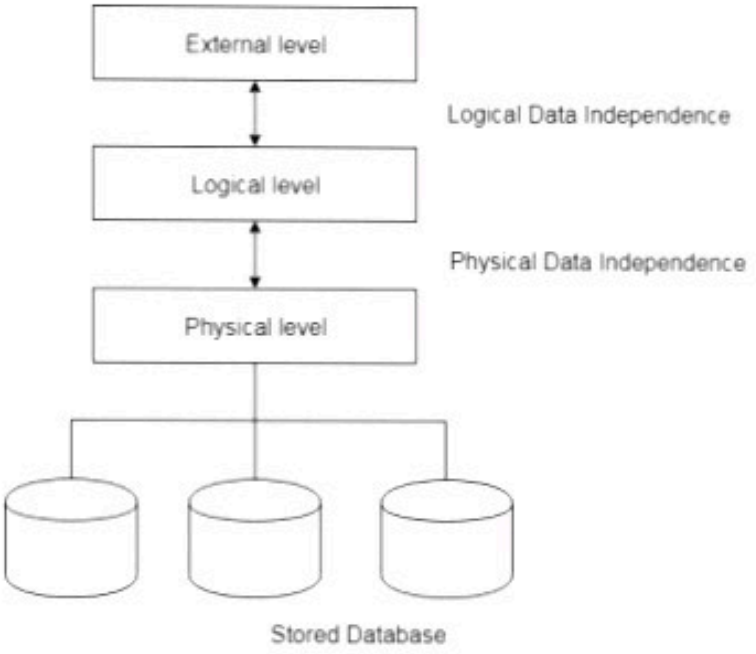


Fig: Data Independence

without changing the external schemas of application programs.

part-B 2)

1.5 DATA MODELS:

A **data model** is a collection of concepts that can be used to describe the structure of a database. By *structure of a database* we mean the data types, relationships and constraints that apply to the data.

Types of Data Models:

- Relational model
- Entity-Relationship data model (mainly for database design)
- Object-based data models (Object-oriented and Object-relational)
- Semi-structured data model (XML)

Relational Model

The relational model uses a collection of tables to represent both data and relationships among the data. Each table has multiple columns, and each column has a unique name. The below table called customer table, shows, for example, that the customer identified by customer-id 100 is named john and lives at 12 anna st. in Chennai and also shows his account number.

Customer_id	Customer_name	Cutomer_street	Customer_city	Account_no
100	John	12 anna st	Chennai	A-101
101	Karthik	3 main st	Chennai	A201
103	Lilly	4 north st	Chennai	A-204

The relational model is an example of a **record-based model**. The relational model is at a lower level abstraction than the E-R model. Database designs are often carried out in the E-R model, and then translated to the relational model.

Entity-Relationship data model

The entity-relationship (E-R) data model, models an enterprise as a collection of *entities* and *relationships*.

> **Entity:** is a "thing" or "object" in the enterprise that is distinguishable from other objects. They are described by a set of *attributes*
>
> **Relationship:** is an association among several entities

The E-R model was developed to facilitate database design. The E-R model is very useful in mapping the meanings and interactions of real world enterprises onto a conceptual schema. The E-R model is represented diagrammatically by an *entity-relationship diagram* as shown below. In the **figure (b) customer** and **account** represents **entities**, the **ellipses** represent **attributes** and **depositor** represents **relationship** among the entities.

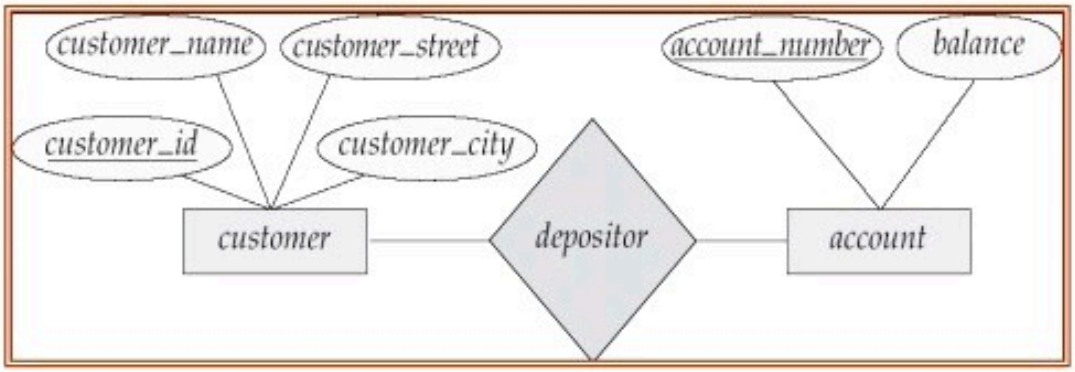


Figure b Object based data models

Object based data models are categorized into object-oriented data model and object-relational data model. The **object–oriented data model** can be seen as extending the E-R model with notions of encapsulation, methods or functions and object identity. The **object-relational model** extends the relational data model by including object orientation and constructs to deal with added data types.

Semi-structured data model (XML)

Extensible markup Language is defined by the WWW Consortium (W3C). It was originally intended as a document markup language and not a database language. It has the abilityto specify new tags, and to create nested tag structures which made XML a great way to exchange **data**, not just documents. XML has become the basis for all new generation data interchange formats. A wide variety of tools is available for parsing, browsing and querying XML documents/data

Hierarchical Model:

In the hierarchical model, data is organized as an inverted tree. Each entity has only one parent but can have several children. At the top of the hierarchy, there is one entity, which is called the root.

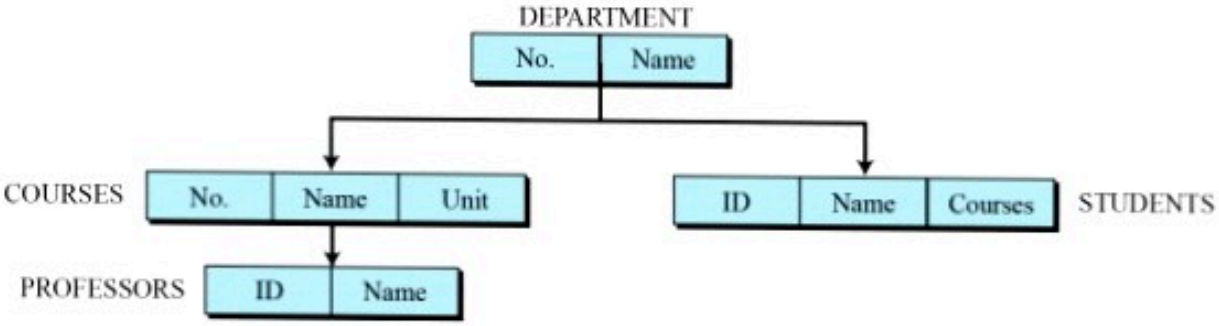


Fig. An example of the hierarchical model representing a university

Network Model:

In the network model, the entities are organized in a graph, in which some entities can be accessed through several paths.

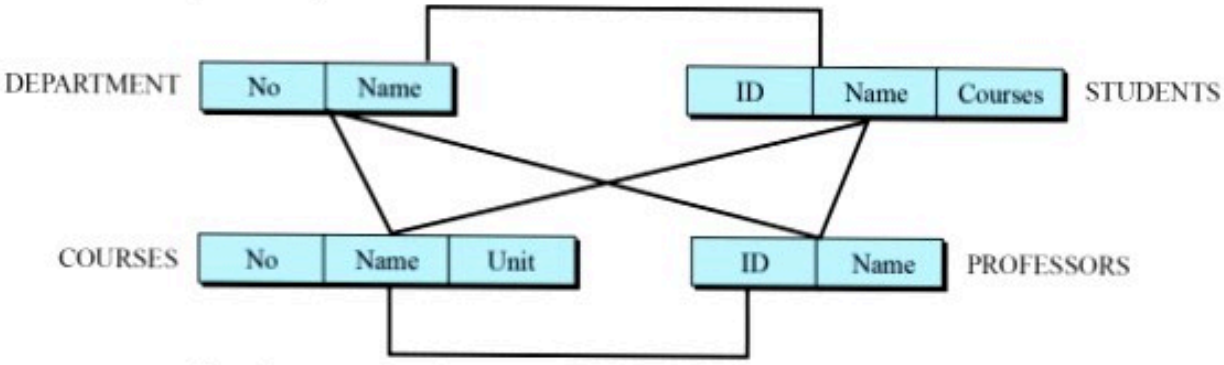


Fig. An example of the Network model representing a university

part-B 3)

Operations in Relational Algebra:

Select, Project, Cartesian Product, Set operations, Join operations, Rename.

1) Select operation:

The select operation selects tuples that satisfy a given predicate. The lower case Greek letter sigma (σ) is used to denote the selection.

⇒ Sample:

Loan no	Branch name	Amount
1	Mianus	900
2	Mianus	1500
3	Round Hill	1300
4	Downtown	2000

σ branchname = "Mianus" (loan)

Loan no	Branch_Name	Amount
1	Mianus	900
2	Mianus	1500

2) Project Operation:

The project operation is a unary operation that returns its argument relation denoted as Pi (π)

π loan no, amount (loan)

Loan no	amount
1	900
2	1500
3.	1300
4.	2000

3) Set operation:

There are several set operations like union, difference, intersection.

Example:

BORROWER RELATION

customer name	loan-no
Adams	16
Curry	93
Hayes	15
Jackson	14

DEPOSITER RELATION

customer name	Account number
Hayes	101
Curry	102
Jones	103
Smith	104

I) UNION

Π customer name (borrower) $\cup \Pi$ (customer name (depositer)

customer_name
Adams
Curry
Hayes
Jackson
Jones
Smith

II) DIFFERENCE

Π customer name (depositor) $- \Pi$ customer name (borrower)

customer name
Jones
Smith

III) INTERSECTION

Π customer name (depositor) $\cap \Pi$ customer name (borrower)

customer name
Hayes
Curry

Rename operation

Rename operation is used to rename the attributes

It is denoted as ρ(rho)

⇒ Example:

$[\rho(\text{rho})(\text{Scholars})(\text{Students})]$

Students:

Id	Name
1	John
2	Emma

Scholars

Id	Name
1	John
2	Emma

1.9 SQL FUNDAMENTALS

Part-B 4)

1.9.1 SQL Standards:

The SQL language has several parts:

- **Data-definition Language (DDL):** the SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.
- **Interactive data-manipulation language (DML):** the SQL DML includes a query language based on both the relational algebra and the tuple relational calculus. It includes also commands to insert tuples into, delete tuples from, and modify tuples in the database.
- **View definition:** the SQL DDL includes commands for defining views

- **Transaction control:** SQL includes commands for specifying the beginning and ending of transactions.

- **timestamp:** a combination of date and time.

1.9.3 Data definition Language:

The SQL data definition language allows specification of not only a set of relations but also information about each relation, including

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

DDL Commands:

1) **Create Table Construct:**

An SQL relation is defined using the **create table** command:

Syntax:

create table *r* (*A*1 *D*1, *A*2 *D*2, ..., *An Dn*,
(integrity-constraint1),
...,
(integrity-constraintk))

where,

r is the name of the relation

Create table products(pid int,supid int,foreign key(supid) references supplier(supid));

2) **Alter table:**
Alter table command is used to add, modify or drop attributes or columns from the table.
Syntax:
alter table <table-name> add/modify <col-name1> <datat ype>,….,<col-namen> <data type>;
alter table <table-name> drop <col-name>;

Example:
alter table emp add year char(3);

3) **Truncate table:**
The truncate table command deletes the rows in the table but retains the table structure.
Syntax:
truncate table <table-name>;
Example:
Truncate table emp;

4) **Drop table**:
Deletes the table from the database.
Syntax:
drop table <table-name>
Example:
drop table emp;

1.9.4 Data Manipulation Language:

It includes also commands to insert tuples into, delete tuples from, and modify tuples in the database.

DML Commands:

Basic Query Structure:

The basic structure of an SQL expression consists of three clauses: **select, from, and where.**

The **select clause** corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.

The **from clause** corresponds to the Cartesian-product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.

The **where clause** corresponds to the selection predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the from clause.

A typical SQL query has the form:

select *A*1, *A*2, ..., *An*
from *r*1, *r*2, ..., *rm*
where *P*

Each Ai represents an attribute
Ri represents a relation and *P* is a predicate.

This query is equivalent to the relational algebra expression.

$$\prod_{A_1, A_2, \ldots, A_n} (\sigma_P (r_1 \times r_2 \times \ldots \times r_m))$$

1) The Select Clause:

The **select** clause list the attributes desired in the result of a query. It corresponds to the projection operation of the relational algebra

Example: find the names of all branches in the *loan* relation:

Sql query:

select *branch_name*
from *loan*

1.9.5 Data Control Language (DCL):

It is used to create roles, permissions, and referential integrity as well it is used to control access to database by securing it.

- GRANT - gives user's access privileges to database
- REVOKE - withdraw access privileges given with the GRANT command

Grant:

Syntax:

grant <privilege list> **on** <relation name or view name> **to** <user/role list>

The privilege list allows the granting of several privileges in one command.

Example 1:

The following **grant** statement grants users U1, U2 and U3 **select** authorization on the account relation.

grant select on account **to** U1,U2,U3

Example 2:

This **grant** statement gives users U1,U2, and U3 update authorization on the amount attribute of the loan relation:

grant update(amount) **on** loan **to** U1,U2,U3

Roles:

Roles can be created in SQL as follows:

Example:

create role teller

Roles can then be granted privileges just as the users can, as illustrated in the below statement.

grant select on account **to** teller

Roles can be assigned to the users, as well as to some other roles, as the below statements show:

grant teller **to** john
create role manager
grant teller **to** manager
grant manager **to** mary

The privilege to grant Privileges:

By default, a user/role that is granted a privilege is not authorized to grant privilege to another user/role. If we wish to grant a privilege and to allow the recipient to pass the privilege on other users, we append **with grant option** clause to the appropriate grant command. For example, ifwish to allow U1 the **select** privilege on branch and allow U1 to grant privilege to others, the query written is

grant select on branch **to** U1 **with grant option**

Revoke :

To revoke an authorization, the revoke statement is used. It takes the form almost identical to that grant format.

revoke <privilege list> **on** <relation name or view name>
from <user/role list> [**restrict | cascade**]

Thus, to revoke the privileges that were granted previously, the queries are written as below:

revoke select on branch **from** U1,U2,U3
revoke update (amount) **on** loan **from** U1,U2,U3

The revocation of a privilege from a user/role may cause other users/roles also to lose privilege. This behavior is called cascading of the revoke. In most database systems, cascading is the default behavior. The revoke statement may alternatively specify **restrict**.

revoke select on branch **from** U1,U2,U3 **restrict**

In this case, the system returns an error if there are any cascading revokes, and does not carry out the revoke action. The following **revoke** statement revokes only the grant option, rather than the actual select privilege.

revoke grant option for select on branch **from U1**

1.9.6 Transaction Control (TCL):

These statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.

- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT

Example:

```
SQL>select * from borrower;
CNAME                LNO
-------------------- ----------
adams                 11
smith                 21
jones                 31
SQL> savepoint a;
```



```
Savepoint created.
SQL> delete from borrower;
3 rows deleted.
SQL> rollback to a;
Rollback complete.
SQL> select * from borrower;
CNAME                LNO
-------------------- ----------
adams                 11
smith                 21
jones                 31
```

KEYS

part-B 5)

What are Keys?

A DBMS key is an attribute or set of an attribute which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables. Keys help you uniquely identify a row in a table by a combination of one or more columns in that table.

Example:

Employee ID	FirstName	LastName
11	Andrew	Johnson
22	Tom	Wood
33	Alex	Hale

Various Keys in Database Management System

DBMS has following seven types of Keys each have their different functionality:

- Super Key
- Primary Key

- Candidate Key
- Alternate Key
- Foreign Key
- Compound Key
- Composite Key
- Surrogate Key

What is the Super key?

A superkey is a group of single or multiple keys which identifies rows in a table. A Super key may have additional attributes that are not needed for unique identification.

Example:

EmpSSN	**EmpNum**	**Empname**
9812345098	AB05	Shown
9876512345	AB06	Roslyn
199937890	AB07	James

In the above-given example, EmpSSN and EmpNum name are superkeys.

What is a Primary Key?

A column or group of columns in a table which helps us to uniquely identifies every row in that table is called a primary key. This DBMS can't be a duplicate. The same value can't appear more than once in the table.

(e) Rules for defining Primary key:

- Two rows can't have the same primary key value
- It must for every row to have a primary key value.
- The primary key field cannot be null.
- The value in a primary key column can never be modified or updated if any foreign key refers to that primary key.

Example:

In the following example, <code>StudID</code> is a Primary Key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

What is the Alternate key?

All the keys which are not primary key are called an alternate key. It is a candidate key which is currently not the primary key. However, A table may have single or multiple choices for the primary key.

Example: In this table.

StudID, Roll No, Email are qualified to become a primary key. But since StudID is the primary key, Roll No, Email becomes the alternative key.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com

What is a Candidate Key?

A super key with no repeated attribute is called candidate key.

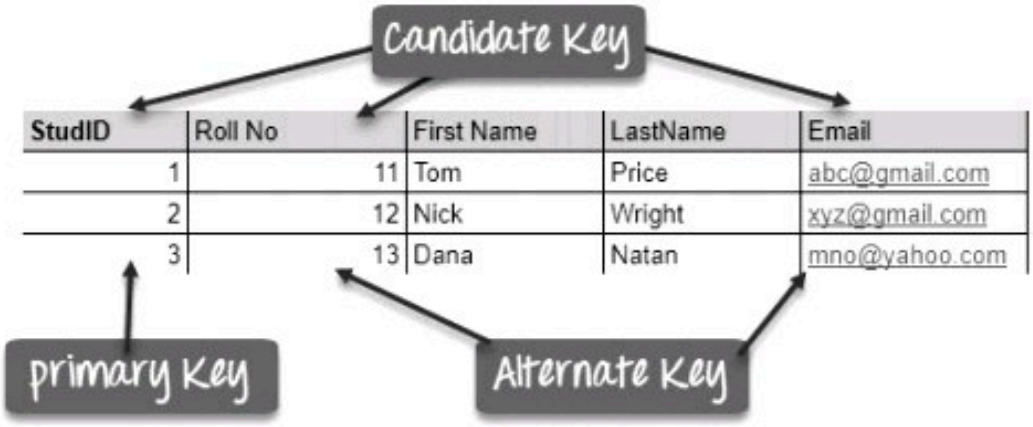
The Primary key should be selected from the candidate keys. Every table must have at least a single candidate key.

Properties of Candidate key:

- It must contain unique values
- Candidate key may have multiple attributes
- Must not contain null values
- It should contain minimum fields to ensure uniqueness
- Uniquely identify each record in a table

Example: In the given table Stud ID, Roll No, and email are candidate keys which help us to uniquely identify the student record in the table.

StudID	Roll No	First Name	LastName	Email
1	11	Tom	Price	abc@gmail.com
2	12	Nick	Wright	xyz@gmail.com
3	13	Dana	Natan	mno@yahoo.com



What is the Foreign key?

A foreign key is a column which is added to create a relationship with another table. Foreign keys help us to maintain data integrity and also allows navigation between two different instances of an entity. Every relationship in the model needs to be supported by a foreign key.

Example:

DeptCode	DeptName
001	Science
002	English
005	Computer

Teacher ID	Fname	Lname
B002	David	Warner
B017	Sara	Joseph
B009	Mike	Brunton

In this example, we have two table, teach and department in a school. However, there is no way to see which search work in which department.

In this table, adding the foreign key in Deptcode to the Teacher name, we can create a relationship between the two tables.

Teacher ID	DeptCode	Fname	Lname
B002	002	David	Warner
B017	002	Sara	Joseph
B009	001	Mike	Brunton

This concept is also known as Referential Integrity.

What is the Compound key?

Compound key has many fields which allow you to uniquely recognize a specific record. It is possible that each column may be not unique by itself within the database. However, when combined with the other column or columns the combination of composite keys become unique.

Example:

OrderNo	PorductID	Product Name	Quantity
B005	JAP102459	Mouse	5
B005	DKT321573	USB	10
B005	OMG446789	LCD Monitor	20
B004	DKT321573	USB	15
B002	OMG446789	Laser Printer	3

In this example, OrderNo and ProductID can't be a primary key as it does not uniquely identify a record. However, a compound key of Order ID and Product ID could be used as it uniquely identified each record.

1.11 DYNAMIC SQL

The dynamic SQL component allows programs to construct and submit SQL queries at run time. In contrast, embedded SQL statements must be completely present at compile time. Using dynamic SQL, programs can create SQL queries as strings at run time and can either have them executed immediately or have them prepared for subsequent use. Preparing dynamic SQL statementcompiles it, and subsequent uses of the prepared statement use the compiled version.

SQL defines standards for embedding dynamic SQL calls in a host language, such as C, as in the following example.

```
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.

advanced sql part B 6)

3) Embedded SQL:

Embedded SQL is a method of combining the computing power of a high-level programming languages. This approach allows developers to execute SQL queries directly within the source code of an application, providing a seamless way to interact with databases.

⇒ Components of Embedded SQL:

i) Embedded SQL Statements:

SQL commands embedded directly into the host program, marked by specific delimiters, usually using keywords like EXEC, SQL.

ii) Host variables:

Variables declared on the host language to interact with SQL statements.

iii) ~~Connecting variables~~ SQL Execution

Establishing the SQL queries such as SELECT, INSERT, UPDATE and DELETE.

iv) Connecting management:

Establishing and managing a connection to the database within the host language.

Advantages:

1. Allows seamless integration of SQL queries
2. Better performance than dynamic SQL
3. Reduces runtime errors
4. Portable across different DBMS

Disadvantages:

1. Harder to debug
2. SQL queries are hard coded

1. ~~Explain basic architecture of a~~

7. Consider the employee database, where the Primary keys are underlined.
employee (empname, Street, city)
works (empname, companyname, Salary)
company (company name, city)
manages (empname, management).
Give an expression in the relational algebra for each request

Answer:

i) Find the names of employees who work in a company located in the same city as their residence.

$[\Pi$ (empname) $(\sigma$ (employee.city = company.city) (employee $\bowtie$ (works $\bowtie$ company)))]

```
SELECT e.empname FROM employee e
JOIN works w ON e.empname = w.empname
JOIN company c ON w.companyname = c.companyname
WHERE e.city = c.city;
```

ii) List the names of employees who work for a company managed by themselves.

```
SELECT w.empname FROM works w
JOIN manages m ON w.empname = m.empname;
```

iii) Retrieve the names of employees who work for all companies located in specific city (ex: New York)

```
SELECT empname FROM works w
WHERE NOT EXISTS
( SELECT C.companyname FROM company c
  WHERE C.city = 'New York' AND NOT EXISTS
  ( SELECT 1 FROM works w2 WHERE w2.empname
    = w.empname AND w2.companyname = C.companyname)
```

iv) Find the employees who earn the highest salary in their respective company

```
SELECT w.empname FROM works W
JOIN (SELECT companyname, MAX(Salary) AS
      max_salary FROM works GROUP BY companyname)
  AS max_Salaries ON w.companyname
      = max_salaries.companyname AND
      w.salary = max_salaries.max_salary;
```

v) List the employees who manage a department and live in same city as the company they manage:

```
SELECT e.empname FROM manages m
JOIN works w ON m.empname = w.empname
JOIN employee e ON w.empname = e.empname
JOIN company C ON w.companyname = C.companyname
WHERE e.city = C.city;
```

8. 9) Cartesian Product:

EMPLOYEE × DEPARTMENT

E.NO	NAME	DOB	GENDER	DCODE	DCODE	DNAME
12345	Hamen	24.3.01	M	201	201	Computer Sc
12345	"	"	"	201	202	INFN SC
12345	"	"	"	201	203	CIVIL
12345	"	"	"	201	204	MECHANICAL
12346	Vini	12.03.01	F	202	201	Compute, Sci
12346	"	"	"	202	202	Infn Sc
12346	"	"	"	202	203	civil
12346	"	"	"	202	204	machanical
12347	Ani	11.01.99	F	202	201	Com Sci
"	"	"	"	202	202	Infn Sc
"	"	"	"	202	203	civil
"	"	"	"	202	204	machanical
12348	Peter	16.02.01	M	204	201	comsci
"	"	"	"	204	202	Infn sc
"	"	"	"	204	203	civil
"	"	"	"	204	204	machanical

2) Equi Join

EMPLOYEE ⋈ Employee.Dcode DEPARTMENT

ENO	NAME	BOB	GENDER	DCODE	DNAME
12345	Hamen	24.3.01	M	201	Computersci
12346	Vini	12.3.01	F	202	Infn Sc
12347	Ani	11.01.99	F	202	Infn Sc
12348	Peter	14.02.01	M	204	Mechanical

3) Left outer Join:

EMPLOYEE ⟕ all EMPLOYEE.DCODE

ENO	NAME	DOB	GENDER	DCODE	DNAME
12345	Hamen	24.3.01	M	201	C.S
12346	Vini	12.3.01	F	202	Infn Sc
12347	Ani	11.1.99	F	202	Infn Sc
12348	Peter	14.2.01	M	204	Mech

4) Right outer Join

EMPLOYEE ⟖ all EMPLOYEE.DCOE

~~same as left outer~~

ENO	NAME	DOB	GENDER	DCODE	DNAME
12345	Hamen	24.3.01	M	201	CS
12346	Vini	12.3.01	F	202	Infn Sc
12347	Ani	11.01.99	F	202	Infn Sc
Null	null	null	null	203	Civil
12348	Peter	14.02.01	M	204	Mech

5) Full outer Join:

EMPLOYEE ⟗ all Employee.Dcode

ENO	NAME	DOB	GENDER	DCODE	DNAME
12345	Hamen	24.3.01	M	201	C.S
12346	Vini	12.3.01	F	202	Infn Sc
12347	Ani	11.1.99	F	202	Infn Sc
12348	Peter	14.2.01	M	204	Mech
Null	null	null	null	203	Civil