DATABASE DESIGN AND MANAGEMENT

INTRODUCTION

Data and information are major resources in an organization. Similarly, the management of data and information is an important business activity in every organization. In the traditional office, every unit or department had its own set of files. The personnel department had a file for each employee. The accounts department also has a file for each employee. In a situation in which a file had to leave one unit to another, there is a problem of misgiving, resistance or hostility. The file system which has been the earlier form of information storage is characterized by various problems. Some of these problems are:

1. **Data Duplication :** Since many data elements might be stored in more than one files and each file organized in a different way, there is a high tendency that the same data are replicated in several files. This duplication of data can be costly in terms of data preparation, storage space, update and processing time, which will also lead to data integrity (inconsistency) problem.
2. **File Maintenance Problems**: Storing complex data in a file system can be more expensive in terms of processing. Every time the value of a given data element changes, that change must be made to every record in which the old values occurs. And too often, the change is made to only one record. Therefore, a specific data element may have different values in different files.
3. **Flexibility Problem:** In a situation where a request for an information needs to be answered. These request may require analysts to restructure existing files.
4. **Insecurity**: Files are mostly kept in cupboards, lockers etc. and because there is no inherent mechanism that help detect when a file is being tempered with or modified, un-authorized users can easily gain access to the office where these files are being kept.

Scanned with CamScanner

Database Approach

The difficulties that arise from using the file-based system have prompted the development of a new approach in managing large amounts of organizational information called the *database approach*. Databases and database technology play an important role in most areas where computers are used, including business, education and medicine. To understand the fundamentals of database systems, we will start by introducing some basic concepts in this area.

What is Database?

Data can be defined as a known fact that can be recorded and have an implicit meaning. In computer science, data is anything in a form suitable for use with a computer. Data can be found in various forms such as text, numbers, figures, tables, diagram, pictures, images, symbols, videos and sounds. Today we live in an era where we need data for all our daily life. In the process of dealing with our day to day data, we interact with different kinds of databases. For instance, a person withdraws money from his bank account on his way to the office, then during his office time, he checks for some details about some products from the internet and in the evening he purchases grocery items from a nearby supermarket. In this case, the person does nothing but interact with different databases for different needs.

A database is a collection of information that is organized so that it can easily be accessed, managed and updated by a computer system. A database system generally provides simultaneous access for different users. It is therefore has to process several user requests (transactions) in parallel and guarantee a distinct level of fault tolerance. A database software should enforce the four requirements which are called the **ACID**-principle/rules:

Atomicity: From the view of a user, all task in a transaction must either be completed as a whole or not at all. A transaction is said to be atomic if it is completed or the transaction failed to occur.

Consistency: A transaction is said to be consistent if it is successful and if a failure occurs, returns all data to its initial state.

Isolation: Every transaction is isolated from all other transactions. This also implies that a transaction can access only data that are part of a consistent state of the database.

Scanned with CamScanner

Durability: If a transaction has been reported to the user as successfully completed, all changes made in the database will survive subsequent hardware or software failures (unless the database is destroyed as a whole). That is, a completed transaction cannot be aborted or discarded even when the DBMS is restarted after a certain failure.

Database Management System

A database management system (DBMS) is a set of software that organizes the storage of data. In large systems, a DBMS allows users and other software to store and retrieve data in a more structured way. A DBMS can be an extremely complex set of software programs that controls the organization, storage and retrieval of data (fields, records and files) in a database. It controls the security and integrity of the database, accepts requests for data from the application program and instructs the operating system to transfer the appropriate data.

ADVANTAGES OF DBMS

1. **Removal of Data Redundancy:** Sometimes, data redundancy cannot be eliminated. However, with DBMS, unnecessary redundancy can be avoided. Another way to put it is that data duplication is controlled or minimized.
2. **Efficient Data Access:** A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently.
3. **Data Consistency and Integrity:** If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. A situation in which there is conflicting information coming from different departments or units is quite objectionable. When the same data elements are scattered or replicated in several units, there is no assurance that if they are updated in one unit, they will be updated in all units. When there is only one entry of a data element, then the problem of inconsistency cannot arise.
4. **Data Security:** DBMS can prevent unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or a subset of it. Many DBMS applications can easily track changes made to tables in the database along with identifying and recording the identity of the person who altered the data.

Scanned with CamScanner

5. **Data Administration:** Centralizing the administration of data can offer significant improvements, minimize redundancy and fine-tuning the storage of the data to make retrieval efficient.
6. **Concurrent Access:** By controlling access to data items, a DBMS can schedules concurrent access to the data in such a manner that users can think of the data as being accessed by only one user at a time.
7. **Reduced Application Development Time**: Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates the quick development of applications.
8. **Enforced Standard:** Centralized planning ensures not only uniformity in the construction of data structures (the format of storing data elements) but guarantees that all necessary standards are met. It is the standardization of the format of data storage that allows different programs to access the database.

TYPES OF DATABASE MANAGEMENT SYSTEMS

The concept of DBMS is usually categorized based on a particular database model. The three (3) major database models are described as follows:

HIERARCHICAL DATABASE MODEL

The hierarchical data model organizes data in a treelike structure. In this model, there is a hierarchy of parent and child data segments. This structure implies that a record can have repeating information, generally in the child data segments. It collects all the instances of a specific record together as a record type. To create links between these record types, the hierarchical model uses parent-child relationships. This is done by using trees, for example, an organization might store information about an employee, such as name, employee number, department, salary. The organization might also store information about an employee's children, such as name and date of birth. The employee and children data form a hierarchy, where the employee data represents the parent segment and the children data represents the child

Scanned with CamScanner

segment. If an employee has three children, then there would be three child segments associated with one employee segment. Note that, In a hierarchical database model the parent-child relationship is one-to-many (1: M) since each node is pointed to/by only one node. This restricts a child segment to having only one parent segment. Hierarchical DBMSs were popular from the late 1960s, with the introduction of IBM's Information Management System (IMS) DBMS, through the 1970s.

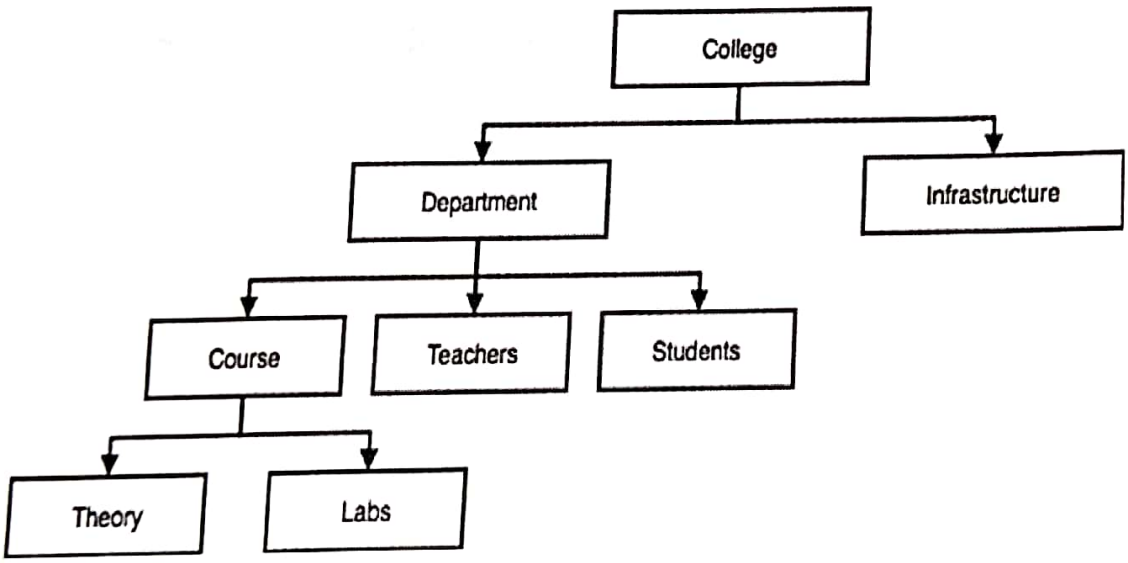


Figure 1: A hierarchical model. Thus there is a one-to-many relationship in this model.

NETWORK DATABASE MODEL

The network database model consists of a more complex relationships structure between the records. It resembles the hierarchical data model with one important exception i.e a child can have more than one parent. Some data were more naturally modelled with more than one parent per child. Therefore, the network model permitted the modelling of a many-to-many relationship in data. Thus, the complete network of relationships is represented by several pairwise sets; in each set, some (one) record type is the owner (at the tail of the network arrow) and one or more record types are members (at the head of the relationship arrow). Usually, the network model tries to relate to many records and accessing is done by relating one of the

Scanned with CamScanner

several paths. Hence this model, therefore allows a many-to-many relationship, although 1:M is permitted.

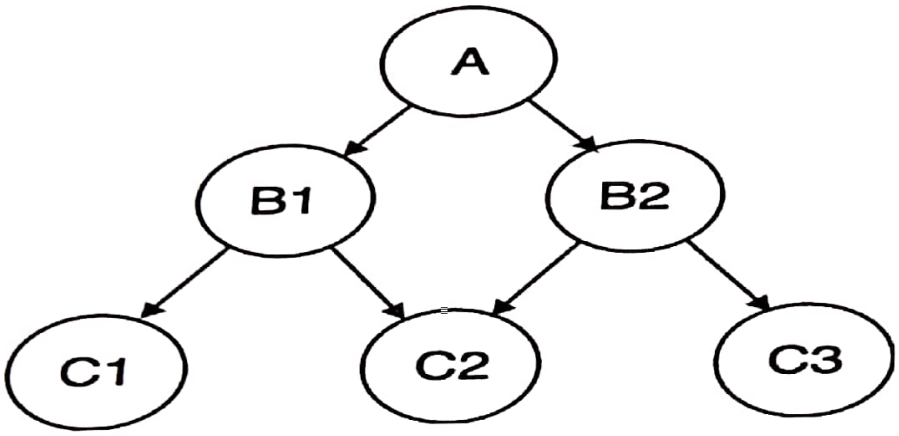


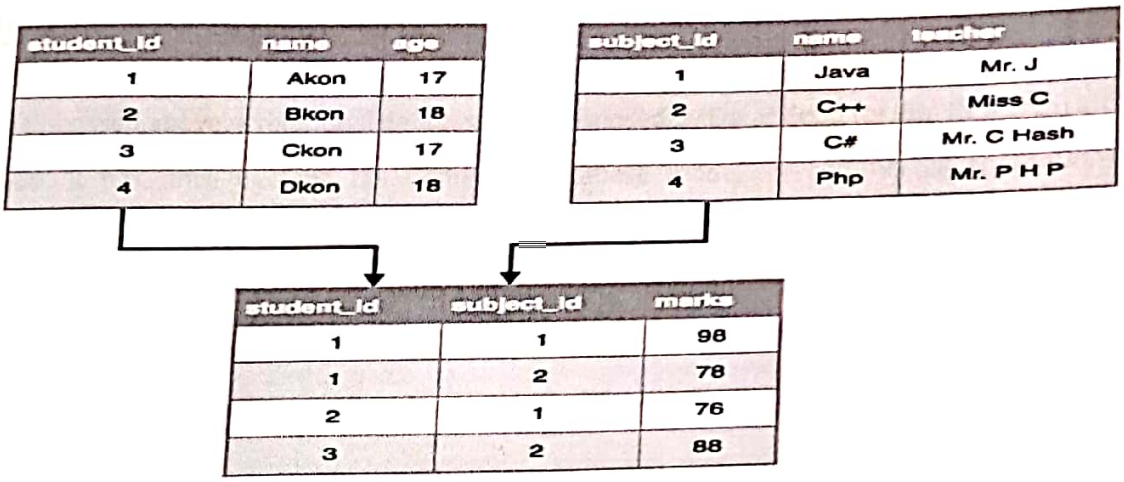
Figure 2: A Network Model. Thus, the element of this model represents a many-to-many relationship.

RELATIONAL DATABASE MODEL

The relational database management system is a database based on the relational model. In this database model, the data and relations between them are organized and presented in tables. A table is a collection of records and each record in a table contains the same fields. Each row of each table describe a record, and each column describes one of the attributes of the record.

Certain fields may be designated as keys (Primary and Foreign Keys), which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables.

Scanned with CamScanner



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

Figure 3: Relational Model to demonstrate a relationship between tables.

OBJECT-ORIENTED MODEL

An Object-Oriented DBMS is the result of combining object-oriented programming principles with database management principles. Object-oriented programming concepts such as encapsulation, polymorphism and inheritance are enforced as well as database management concepts such as atomicity, consistency, integrity and support for an ad-hoc query language. Object-oriented DBMSs add database functionality to object programming languages.

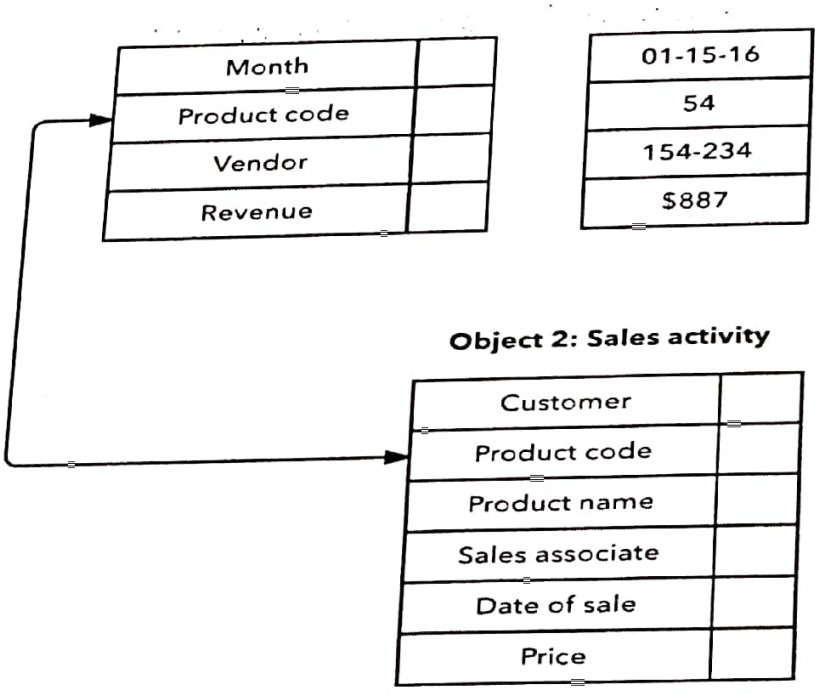


Figure 4: An Object-Oriented Database Model

Scanned with CamScanner

Relational Database Model in Details

As discussed above, a relational database was proposed by Edgar Codd (of IBM Research) around 1969. It has since become the dominant database model for commercial applications (in comparison with other database models such as hierarchical, network and object models). Today, there are many commercial Relational Database Management System (RDBMS), such as Oracle, IBM DB2 and Microsoft SQL Server. There are also many free and open-source RDBMS, such as MySQL, mSQL (mini-SQL) and the embedded JavaDB (Apache Derby).

A relational database system contains one or more objects called tables. The data or information for the database are stored in these tables. Tables are uniquely identified by their names and are comprised of columns and rows. Columns contain the column name, data type, and any other attributes for the column. A database table is similar to a spreadsheet. However, the relationships that can be created among the tables enable a relational database to efficiently store a huge amount of data and effectively retrieve selected data.

Relational Database Design Process

Database design is more art than science, as you have to make many decisions. Databases are usually customized to suit a particular application. No two customized applications are alike, and hence, no two databases are alike. Guidelines (usually in terms of what not to do instead of what to do) are provided in making these design decision, but the choices ultimately rest on the designer.

Step 1: Define the purpose of the database (Requirement Analysis): Gather the requirements and define the objective of your database, e.g. drafting out the sample input forms, queries and reports, often helps.

Step 2: Gather data, organize in tables and specify the primary keys: Once you have decided on the purpose of the database, gather the data that is needed to be stored in the

Scanned with CamScanner

database. Divide the data into subject-based tables. Choose one column (or a few columns) as the so-called primary key, which uniquely identify each of the rows.

Primary Key

In the relational model, a table cannot contain duplicate rows, because that would create ambiguities in retrieval. To ensure uniqueness, each table should have a column (or a set of columns), called a primary key that uniquely identifies every record of the table. For example, a unique number of customerID can be used as the primary key for the Customers table; productCode for Products table; isbn for Books table. A primary key is called a simple key if it is a single column; it is called a composite key if it is made up of several columns.

Most RDBMS build an index on the primary key to facilitate fast search and retrieval. The primary key is also used to reference other tables (to be elaborated later). You have to decide which column(s) is to be used for the primary key. The decision may not be straight forward but the primary key shall have these properties:

The values of the primary key shall be unique (i.e., no duplicate value). For example, customerName may not be appropriate to be used as the primary key for the Customers table, as there could be two customers with the same name. The primary key shall always have value. In other words, it shall not contain NULL. The Primary key often uses integer (or number) type. But it could also be other types, such as texts. However, it is best to use a numeric column as the primary key for efficiency.

Let's illustrate with an example: a table customer contains columns lastName, firstName, phoneNumber, address, city, state, zipCode. The candidates for primary key are name=(lastName, firstName), phoneNumber, Address1=(address, city, state), Address1=(address, zipCode). Name may not be unique. Phone number and address may change. Hence, it is better to create a fact-less auto-increment number, says customerID, as the primary key.

Scanned with CamScanner

Step 3: Create Relationships among Tables: A database consisting of independent and unrelated tables serves little purpose (you may consider using a spreadsheet instead). The power of a relational database lies in the relationship that can be defined between tables. The most crucial aspect in designing a relational database is to identify the relationships among tables. The types of relationship include:

1. One-to-Many
2. Many-to-Many
3. One-to-One

One-to-Many

In a "class roster" database, a teacher may teach zero or more classes, while a class is taught by one (and only one) teacher. In a "company" database, a manager manages zero or more employees, while an employee is managed by one (and only one) manager. In a "product sales" database, a customer may place many orders; while an order is placed by one particular customer. This kind of relationship is known as one-to-many.

A one-to-many relationship cannot be represented in a single table. For example, in a "class roster" database, we may begin with a table called Teachers, which stores information about teachers (such as name, office, phone and email). To store the classes taught by each teacher, we could create columns class1, class2, class3, but faces a problem immediately on how many columns to create. On the other hand, if we begin with a table called Classes, which stores information about a class (courseCode, dayOfWeek, timeStart and timeEnd); we could create additional columns to store information about the (one) teacher (such as name, office, phone and email). However, since a teacher may teach many classes, its data would be duplicated in many rows in table Classes.

To support a one-to-many relationship, we need to design two tables: a table Classes to store information about the classes with classID as the primary key; and a table Teachers to store

Scanned with CamScanner

information about teachers with teacherID as the primary key. We can then create the one-to-many relationship by storing the primary key of the table Teacher (i.e., teacherID) (the "one"-end or the parent table) in the table classes (the "many"-end or the child table).

The column teacherID in the child table Classes is known as the foreign key. A foreign key of a child table is a primary key of a parent table, used to reference the parent table. Take note that for every value in the parent table, there could be zero, one, or more rows in the child table. For every value in the child table, there is one and only one row in the parent table.

Many-to-Many

In a "product sales" database, a customer's order may contain one or more products; and a product can appear in many orders. In a "bookstore" database, a book is written by one or more authors; while an author may write zero or more books. This kind of relationship is known as many-to-many.

Let's illustrate with a "product sales" database. We begin with two tables: Products and Orders. The table products contain information about the products (such as name, description and quantityInStock) with productID as its primary key. The table orders contain customer's orders (customerID, dateOrdered, dateRequired and status). Again, we cannot store the items ordered inside the Orders table, as we do not know how many columns to reserve for the items. We also cannot store the order information in the Products table.

To support a many-to-many relationship, we need to create a third table (known as a junction table), says OrderDetails (or OrderLines), where each row represents an item of a particular order. For the OrderDetails table, the primary key consists of two columns: orderID and productID, that uniquely identify each row. The columns orderID and productID in OrderDetails table are used to reference Orders and Products tables, hence, they are also the foreign keys in the OrderDetails table.

Scanned with CamScanner

One-to-One

In a "product sales" database, a product may have optional supplementary information such as image, moreDescription and comment. Keeping them inside the Products table results in many empty spaces (in those records without these optional data). Furthermore, these large data may degrade the performance of the database. Instead, we can create another table (ProductDetails, ProductLines or ProductExtras) to store the optional data. A record will only be created for those products with optional data. The two tables, Products and ProductDetails, exhibit a one-to-one relationship. That is, for every row in the parent table, there is at most one row (possibly zero) in the child table. The same column productID should be used as the primary key for both tables.

Some databases limit the number of columns that can be created inside a table. You could use a one-to-one relationship to split the data into two tables. A One-to-one relationship is also useful for storing certain sensitive data in a secure table, while the non-sensitive ones in the main table.

Step 4: Refine & Normalize the Design: For example, adding more columns, create a new table for optional data using a one-to-one relationship, split a large table into two smaller tables and checking all the Integrity rules involved in database design such as:

Entity Integrity Rule: The primary key cannot contain NULL. Otherwise, it cannot uniquely identify the row. For a composite key made up of several columns, none of the columns can contain NULL. Most of the RDBMS check and enforce this rule.

Referential Integrity Rule: Each foreign key value must be matched to a primary key value in the table referenced (or parent table). You can insert a row with a foreign key in the child table only if the value exists in the parent table. If the value of the key changes in the parent table (e.g., the row updated or deleted), all rows with this foreign key in the child table(s) must be handled accordingly. You could either (a) disallow the changes (b) cascade the change (or delete the records) in the child tables accordingly (c) set the key value in the child tables to NULL. Most RDBMS can be set up to perform the check and ensure the referential integrity in a specified manner.

Scanned with CamScanner

Step 5: Enter data

This involves the process of entering data into the database.

Functional Dependency

Functional dependency *FD* is a set of constraints between two attributes in a relation. Functional dependency says that if two tuples have the same values for attributes A1, A2,..., An, then those two tuples must have the same values for attributes B1, B2, ..., Bn.

Functional dependency is represented by an arrow sign → that is, X→Y, where X functionally determines Y. The left-hand side attributes determine the values of attributes on the right-hand side.

Armstrong's Axioms

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 a 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.

Trivial Functional Dependency

Trivial – If a functional dependency *FD* X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.

Non-trivial – If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.

Completely non-trivial – If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

Scanned with CamScanner

Normalization

If a database design is not perfect, it may contain anomalies, which are like a bad dream for any database administrator. Managing a database with anomalies is next to impossible.

Update anomalies – If data items are scattered and are not linked to each other properly, then it could lead to strange situations. For example, when we try to update one data item having its copies scattered over several places, a few instances get updated properly while a few others are left with old values. Such instances leave the database in an inconsistent state.

Deletion anomalies – we tried to delete a record, but parts of it are left undeleted because of unawareness, the data is also saved somewhere else.

Insert anomalies – we tried to insert data in a record that does not exist at all.

Normalization is a method to remove all these anomalies and bring the database to a consistent state.

First Normal Form

First Normal Form is defined in the definition of relations *tables* itself. This rule defines that all the attributes in a relation must have atomic domains. The values in an atomic domain are indivisible units.

Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

We re-arrange the relation table as below, to convert it to First Normal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

Each attribute must contain only a single value from its pre-defined domain.

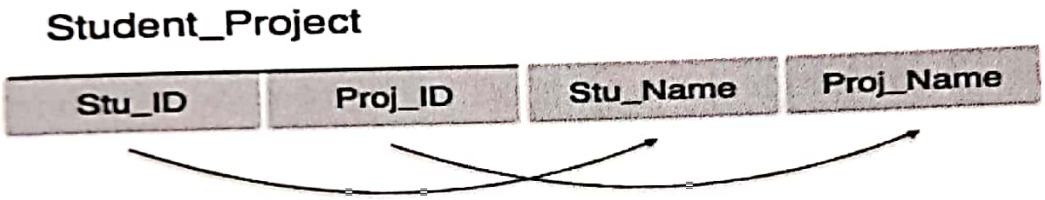
Scanned with CamScanner

Second Normal Form

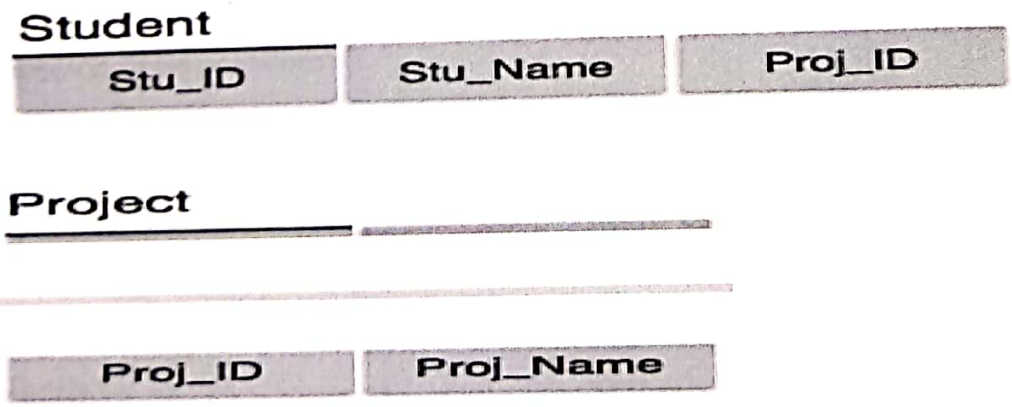
Before we learn about the second normal form, we need to understand the following.

Prime attribute – an attribute, which is a part of the prime-key, is known as a prime attribute.

Non-prime attribute – an attribute, which is not a part of the prime-key, is said to be a non-prime attribute. If we follow the second normal form, then every non-prime attribute should be fully functionally dependent on the prime key attribute. That is, if X → A holds, then there should not be any proper subset Y of X, for which Y → A also holds true.



We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attributes individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in the Second Normal Form.



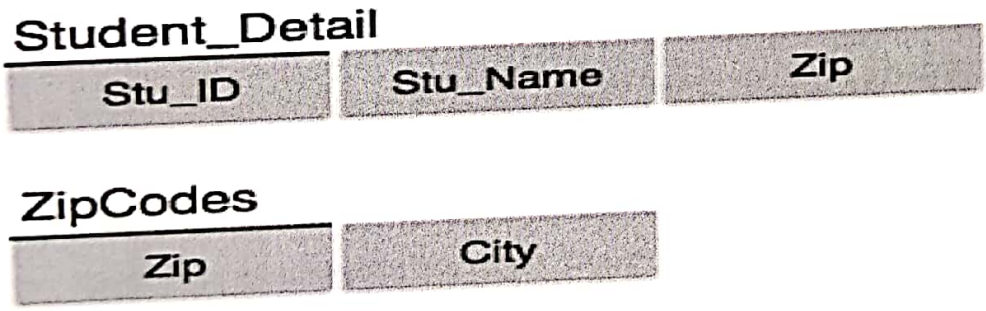
We broke the relation in two as depicted in the above picture. So there exists no partial dependency.

Scanned with CamScanner

Third Normal Form

For a relation to be in Third Normal Form, it must be in Second Normal Form and the following must satisfy –No non-prime attribute is transitively dependent on a prime key attribute. For any non-trivial functional dependency, X → A, then either – X is a superkey or, A is a prime attribute.

We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City so there exists **transitive dependency**. To bring this relation into a third normal form, we break the relation into two relations as follows.



Boyce-Codd Normal Form

Boyce-Codd Normal Form *BCNF* is an extension of the Third Normal Form on strict terms. BCNF states that for any non-trivial functional dependency, X → A, X must be a super-key. In the above image, Stu_ID is the super-key in the relation Student_Detail and Zip is the super-key in the relation ZipCodes. So, Stu_ID → Stu_Name, Zip and Zip → City confirm that both the relations are in BCNF.

Introduction to SQL

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is

Scanned with CamScanner

the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc

SQL Commands

The standard SQL commands to interact with relational databases are CREATE, SELECT, INSERT, UPDATE, DELETE and DROP. These commands can be classified into the following groups based on their nature.

DDL - Data Definition Language

SN.	Command & Description
1	**CREATE** Creates a new table, a view of a table, or other objects in the database.
2	**ALTER** Modifies an existing database object, such as a table.
3	**DROP** Deletes an entire table, a view of a table or other objects in the database.

DML - Data Manipulation Language

SN.	Command & Description
1	**SELECT** Retrieves certain records from one or more tables.
2	**INSERT** Creates a record.

Scanned with CamScanner

3	**UPDATE** Modifies records.
4	**DELETE** Deletes records.

DCL - Data Control Language

SN.	Command & Description
1	**GRANT** Gives a privilege to a user.
2	**REVOKE** Takes back privileges granted from a user.

Statements in SQL

1. SELECT statement

The SELECT statement is used to view the desired information in a relational table. The general form for a SELECT statement, to retrieve all of the rows in a table is:

SELECT ColumnName, ColumnName,

FROM TableName;

Scanned with CamScanner

To get all columns of a table without typing all column names, use:

```
SELECT * FROM TableName;
```

Each DBMS and database software has different methods for logging in to the database and entering SQL commands.

Example: Consider the table below:

StudentsTable

ADMNO	SURNAME	OTHERNAME	DEPARTMENT	ENTRYYEAR
070210001	ABDULLAHI	AMINU	BIOLOGY	2007
070210004	SULEIMAN	MARYAM	PHYSICS	2007
070210201	AHMAD	MURTALA	MATHS	2007
080210001	SANI	AISHATU	MATHS	2008
080210301	DAHIRU	BELLO	ECONOMICS	2008
080210555	UMAR	FAHAD	MATHS	2009

Now, let's say you want to write a query that will display the SURNAME, OTHERNAME and DEPARTMENT of each Student. The SELECT statement will be written as:

```
SELECT SURNAME, OTHERNAME, DEPARTMENT
FROM StudentsTable;
```

The following is the results of your query of the database:

SURNAME	OTHERNAME	DEPARTMENT
ABDULLAHI	AMINU	BIOLOGY
SULEIMAN	MARYAM	PHYSICS
AHMAD	MURTALA	MATHS
SANI	AISHATU	MATHS
DAHIRU	BELLO	ECONOMICS
UMAR	FAHAD	MATHS

RELATIONAL OPERATORS

There are six Relational Operators in SQL.

=	Equal
<> or !=	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal To
>=	Greater Than or Equal To

Scanned with CamScanner

The WHERE clause is used to specify that only certain rows of the table are displayed, based on the criteria described in that WHERE clause.

2. CONDITIONAL SELECTION Statement

To further discuss the SELECT statement, let's look at a new example table below:

EmployeeStatisticsTable

EMPLOYEEIDNO	SALARY(N)	BENEFITS(N)	POSITION
010	75000	15000	HOD
105	65000	15000	HOD
152	60000	15000	HOD
215	60000	12500	LECTURER
244	50000	12000	LECTURER
300	45000	10000	LECTURER
335	40000	10000	LABOURER
400	32000	7500	LABOURER
441	28000	7500	MESSENGER

If you wanted to see only the EMPLOYEEIDNO's of those making at or over N50,000. The Query will be written as follows:

```
SELECT EMPLOYEEIDNO
FROM EMPLOYEESTATISTICSTABLE
WHERE SALARY >= 50000;
```

Notice that the >= (greater than or equal to) sign is used, as we wanted to see those who made greater than N50000 or equal to N50000 listed together. This displays:

Scanned with CamScanner

EMPLOYEEIDNO
010
105
152
215
244

The WHERE description, SALARY >= 50000 is known as a condition (an operation which evaluates to True or False). The same can be done for text columns.

For example, the following statement

```
SELECT EMPLOYEEIDNO
FROM   EMPLOYEESTATISTICSTABLE
WHERE  POSITION = 'HOD';
```

This displays only the ID Numbers of all HODs. Generally, with text columns, stick to equal to or not equal to, and make sure that any text that appears in the statement is surrounded by single quotes (').

3. MORE COMPLEX CONDITIONAL SELECTION: COMPOUND CONDITIONS / LOGICAL OPERATORS

The *AND* operator joins two or more conditions and displays a row only if that row's data satisfies ALL conditions listed (i.e. all conditions hold true). For example, to display all staff making over N40, 000 use:

```
SELECT EMPLOYEEIDNO
FROM EMPLOYEESTATISTICSTABLE
WHERE SALARY > 40000 AND POSITION = 'LABOURER';
```

The *OR* operator joins two or more conditions but returns a row if ANY of the conditions listed hold true. To see all those who make less than N40, 000 or have less than N10, 000 in BENEFITs, listed together, use the following query:

```
SELECT EMPLOYEEIDNO
FROM EMPLOYEESTATISTICSTABLE
WHERE SALARY < 40000 OR BENEFITS < 10000;
```


Scanned with CamScanner

AND & OR can be combined, for example:

```
SELECT EMPLOYEEIDNO
FROM EMPLOYEESTATISTICSTABLE
WHERE POSITION = 'HOD' AND SALARY > 60000 OR BENEFITS > 12000;
```

First, SQL finds the rows where the salary is greater than N60,000 and the position column is equal to HOD, then taking this new list of rows, SQL then sees if any of these rows satisfied the previous AND condition or the condition that the BENEFITs column is greater then N12,000. Subsequently, SQL only displays this second new list of rows, keeping in mind that anyone with BENEFITs over N12,000 will be included as the OR operator includes a row if either resulting condition is True. Also, note that the AND operation is done first. Mathematically, SQL evaluates all of the conditions, then evaluates the AND "pairs" and then evaluates the OR's (where both operators evaluate left to right).

To perform OR's before AND's, for example, if you wish to see a list of employees making a large salary (>N40, 000) or have a large benefits package (>N10, 000), and that happens to be a LECTURER, use parentheses:

```
SELECT EMPLOYEEIDNO
FROM EMPLOYEESTATISTICSTABLE
WHERE POSITION = 'LECTURER' AND (SALARY > 40000 OR BENEFITS > 10000);
```

4. IN & BETWEEN Statements

An easier method of using compound conditions is to use IN or BETWEEN statement. For example, if you want to list all managers and staff:

```
SELECT EMPLOYEEIDNO
FROM EMPLOYEESTATISTICSTABLE
WHERE POSITION IN ('HOD','LECTURER');
```

To list those making greater than or equal to N40,000, but less than or equal to N50,000, use:

Scanned with CamScanner

```
SELECT EMPLOYEEIDNO
FROM EMPLOYEESTATISTICSTABLE
WHERE SALARY BETWEEN 40000 AND 50000;
```

To list everyone not in this range, try:

```
SELECT EMPLOYEEIDNO
FROM EMPLOYEESTATISTICSTABLE
WHERE SALARY NOT BETWEEN 40000 AND 50000;
```

Similarly, NOT IN lists all rows excluded from the IN list.

Additionally, NOT's can be thrown in with AND's & OR's, except that NOT is a unary operator (evaluates one condition, reversing its value, whereas, AND's & OR's evaluate two conditions) and that all NOT's are performed before any AND's or OR's.

SQL ORDER OF LOGICAL OPERATIONS (EACH OPERATES FROM LEFT TO RIGHT)

NOT

AND

OR

5. Using LIKE statement

Look at the StudentsTable, and say you wanted to see all people whose last names started with "L"; try this:

```
SELECT ADMNO
FROM STUDENTSTABLE
WHERE SURNAME LIKE 'L%';
```

The percent sign (%) is used to represent any possible character (number, letter or punctuation) or set of characters that might appear after the "L". To find those people with LastName's ending in "L", use '%L', or if you wanted the "L" in the middle of the word, try '%L%'. The '%' can be used for any characters in the same position relative to the given characters. NOT LIKE displays rows not fitting the given description. Other possibilities of using LIKE, or any of these discussed conditionals are available, though it depends on what DBMS you are using.

Scanned with CamScanner

6. Using DELETE statement

The DELETE statement can be used to delete records in a database.

JOIN Keyword

The JOIN keyword is used in SQL statement to query data from two or more tables, based on the relationship between certain columns in these tables. Tables in a database are often related to each other with keys. A primary key is a column (or combination of columns) with a unique value for each row. Each **primary key** value must be unique within the table. The purpose is to bind data together across tables without repeating all the data in every table. Similarly, a **foreign key** is a column in a table where that column is a primary key of another table, which means that any data in a foreign key column must have corresponding data in the other table where that column is the primary key. In DBMS-speak, this correspondence is known as **referential integrity**. In this section, we will only discuss inner joins, and equijoins, as they are the most useful. Good database design suggests that each table lists data only about a single entity, and detailed information can be obtained in a relational database, by using additional tables, and by using a join. First, take a look at these example tables:

AntiqueOwners

OWNERID	OWNERLASTNAME	OWNERFIRSTNAME
01	Bello	Hassan
02	Aminu	Usman
15	Suleiman	Bello
21	Kabiru	Maryam
50	Buhari	Yahaya

Scanned with CamScanner

Orders

OWNERID	ITEMDESIRED
2	Table
2	Desk
21	Chair
15	Mirror

Antiques

SELLERID	BUYERID	ITEM
1	50	Bed
2	15	Table
15	2	Chair
21	50	Mirror
50	1	Desk
1	21	Cabinet
2	21	Coffee Table
15	50	Chair
1	15	Jewelry Box
2	21	Pottery
21	2	Bookcase
50	1	Plant Stand

In the ANTIQUEOWNERS table, the OWNERID column uniquely identifies that row. This means two things: no two rows can have the same OWNERID, and, even if two owners have the same first and last names, the OWNERID column ensures that the two owners will not be confused with each other, because the unique OWNERID column will be used throughout the database to track the owners, rather than the names.

Scanned with CamScanner

Similarly, in the Antiques table, both the BUYERID and SELLERID are foreign keys to the primary key of the ANTIQUEOWNERs table (OWNERID; for purposes of argument, one has to be an Antique Owner before one can buy or sell any items), as, in both tables, the ID rows are used to identify the owners or buyers and sellers, and that the OWNERID is the primary key of the ANTIQUEOWNERs table. In other words, all of this "ID" data is used to refer to the owners, buyers or sellers of antiques themselves without having to use the actual names.

PERFORMING A JOIN

To find the names of those who bought a chair, use the following query:

```
SELECT OWNERLASTNAME, OWNERFIRSTNAME
FROM ANTIQUEOWNERS, ANTIQUES
WHERE BUYERID = OWNERID AND ITEM = 'Chair';
```

Note the following about this query...notice that both tables involved in the relation are listed in the FROM clause of the statement. In the WHERE clause, first notice that the ITEM = 'Chair' part restricts the listing to those who have bought (and in this example, thereby owns) a chair. Secondly, notice how the ID columns are related from one table to the next by use of the BUYERID = OWNERID clause. Only where ID's match across tables and the item purchased is a chair (because of the AND), will the names from the ANTIQUEOWNERs table be listed. **Because the joining condition used an equal sign, this join is called an equijoin.** The result of this query is two names: Aminu, Usman & Buhari, Yahaya.

DISTINCT Keyword

Let's assume that you want to list the ID and names of only those people who have sold an antique. Obviously, you want a list where each seller is only listed once--you don't want to know how many antiques a person sold, just the fact that this person sold one This means that you will need to tell SQL to eliminate duplicate sales rows, and just list each person only once. To do this, use the DISTINCT keyword.

Scanned with CamScanner

First, we will need an equijoin to the AntiqueOwners table to get the detailed data of the person's LastName and FirstName. However, keep in mind that since the SellerID column in the Antiques table is a foreign key to the AntiqueOwners table, a seller will only be listed if there is a row in the AntiqueOwners table listing the ID and names. We also want to eliminate multiple occurences of the SellerID in our listing, so we use DISTINCT on the column where the repeats may occur. To throw in one more twist, we will also want the list alphabetized by LastName, then by FirstName (on a LastName tie). Thus, we will use the ORDER BY clause:

```
SELECT  DISTINCT SELLERID, OWNERLASTNAME, OWNERFIRSTNAME
FROM  ANTIQUES, ANTIQUEOWNERS
WHERE  SELLERID = OWNERID
ORDER  BY OWNERLASTNAME, OWNERFIRSTNAME;
```

In this example, since everyone has sold an item, we will get a listing of all of the owners, in alphabetical order by last name. For future reference (and in case anyone asks), this type of join is considered to be in the category of **inner joins**.

Aggregate Functions

SQL offers a variety of aggregate functions which return a value based on a number of inputs. Example of aggregate functions are SUM, AVG, MAX, MIN, and COUNT etc. They are called aggregate functions because they summarize the results of a query, rather than listing all of the rows.

SUM () gives the total of all the rows, satisfying any conditions, of the given column, where the given column is numeric.

AVG () gives the average of the given column.

MAX () gives the largest figure in the given column.

MIN () gives the smallest figure in the given column.

COUNT (*) gives the number of rows satisfying the conditions.

Examples:

Scanned with CamScanner

```
SELECT SUM (SALARY), AVG (SALARY)
FROM EMPLOYEESTATISTICSTABLE;
```

This query shows the total of all salaries in the table and the average salary of all of the entries in the table.

```
SELECT MIN (BENEFITS)
FROM EMPLOYEESTATISTICSTABLE
WHERE POSITION = 'LECTURER';
```

This query gives the smallest figure of the Benefits column, of the employees who are LECTURERs, which is 10000.

```
SELECT COUNT (*)
FROM EMPLOYEESTATISTICSTABLE
WHERE POSITION = 'LABORER';
```

This query tells you how many employees have LABORER status (2).

Altering Tables in SQL

Tables can be modified in SQL using the ALTER TABLE statement as follows:

1- **Adding a column:**

```
ALTER TABLE <table name>
ADD <column name> <data type>
```

2- **Dropping a column:**

```
ALTER TABLE <table name>
DROP <column name>
```

3- **Modifying a column:**

```
ALTER TABLE <table name>
MODIFY <column name> <data type>
```

Example:

1. Adding a column

   Assuming that we decide to add a column to the Staff table to specify the rank of the s

   This will be written as:

```
ALTER TABLE Staff  ADD (Rank char (10) Not Null);
```

Scanned with CamScanner

Adding Data into Tables

After tables are successfully created, data or records can be entered into the existing table(s) using the INSERT INTO statement.

Rows/records can be inserted into Staff table created above as follows:

```
INSERT INTO STAFF VALUES (101,'Mr','Aminu','Biology','Lecturer');
INSERT INTO STAFF VALUES (102,'Mrs','Aisha','Physics','Lecturer I');
```

This inserts the data into the table, as 2 new rows, column-by-column, in the pre-defined order.

. Removing TABLE in SQL

Tables can be destroyed (removed from the database) in SQL using the DROP TABLE statement.

Example:

```
DROP TABLE Staff;
```

This statement will delete/remove the entire staff table from your database.

4. Deleting data or record from a TABLE in SQL

DELETE queries allow you to delete whole rows (records), NOT individual field values of a table.

Example:

```
DELETE FROM Staff
WHERE DEPT = 'BIOLOGY';
```

This deletes all the rows that contain Biology in the table.

5. Updating Data in TABLES

Rows or records in Tables can also be changed using the UPDATE STATEMENT.

Example:

Let's try to change the rank of a staff in the staff Table. This can be written as follows:

```
UPDATE STAFF SET RANK = 'SENIOR LECTURER' WHERE STAFFCODE = 101;
```

Or

Scanned with CamScanner

```
UPDATE STAFF
SET RANK = 'SENIOR LECTURER'
WHERE STAFFCODE = 101;
```

This sets the rank of the lecturer with the specified ID to the new rank specified i.e Senior Lecturer.

Scanned with CamScanner