# SNS COLLEGE OF TECHNOLOGY

**Coimbatore-35**
**An Autonomous Institution**

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A++' Grade Approved
by AICTE, New Delhi & Affiliated to Anna University, Chennai

## DEPARTMENT OF COMPUTER APPLICATIONS

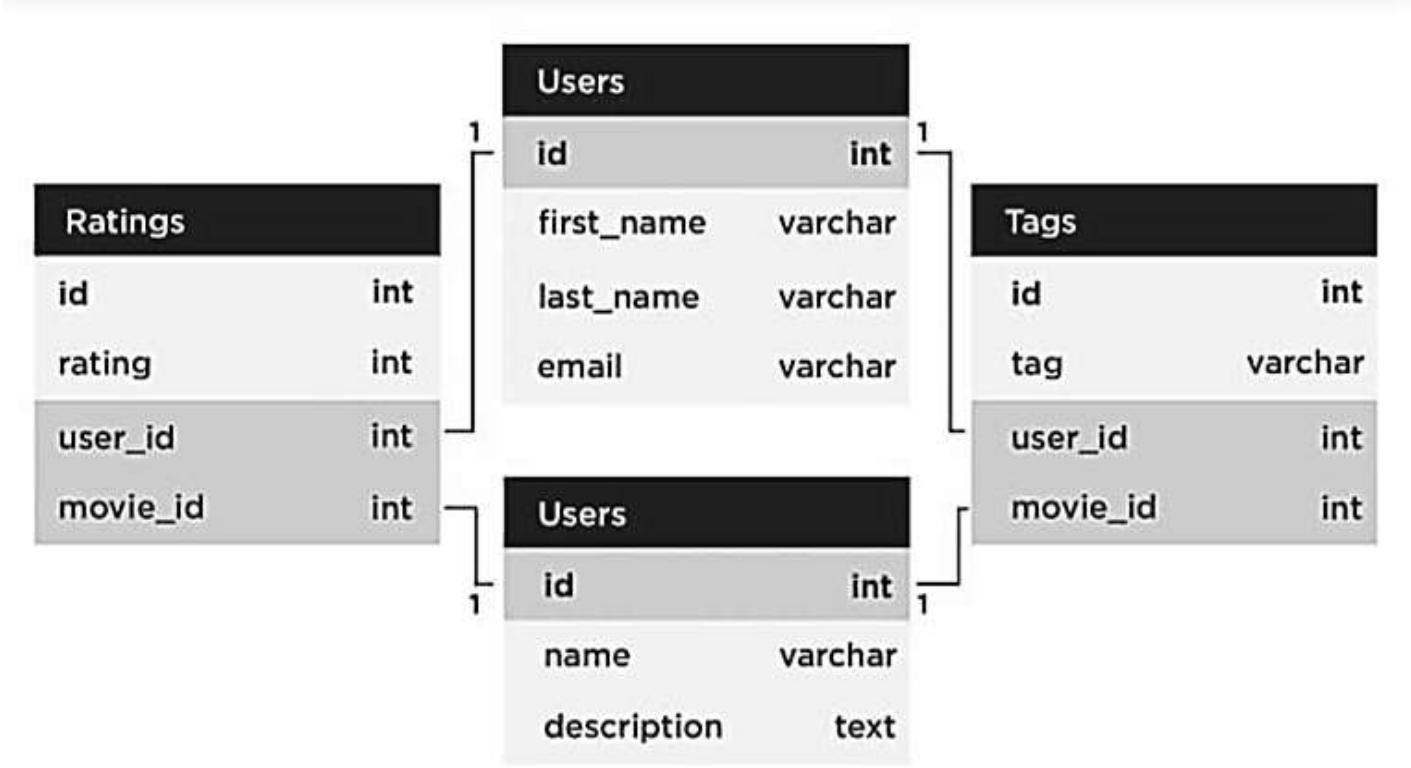
**19CAE730 – Fundamentals of NOSQL database System**
II YEAR III SEM

UNIT I - **INTRODUCTION**

TOPIC 3 &4 & 5: The Value of Relational Databases, Getting at Persistent Data, Concurrency, Integration

# The Value of Relational Databases

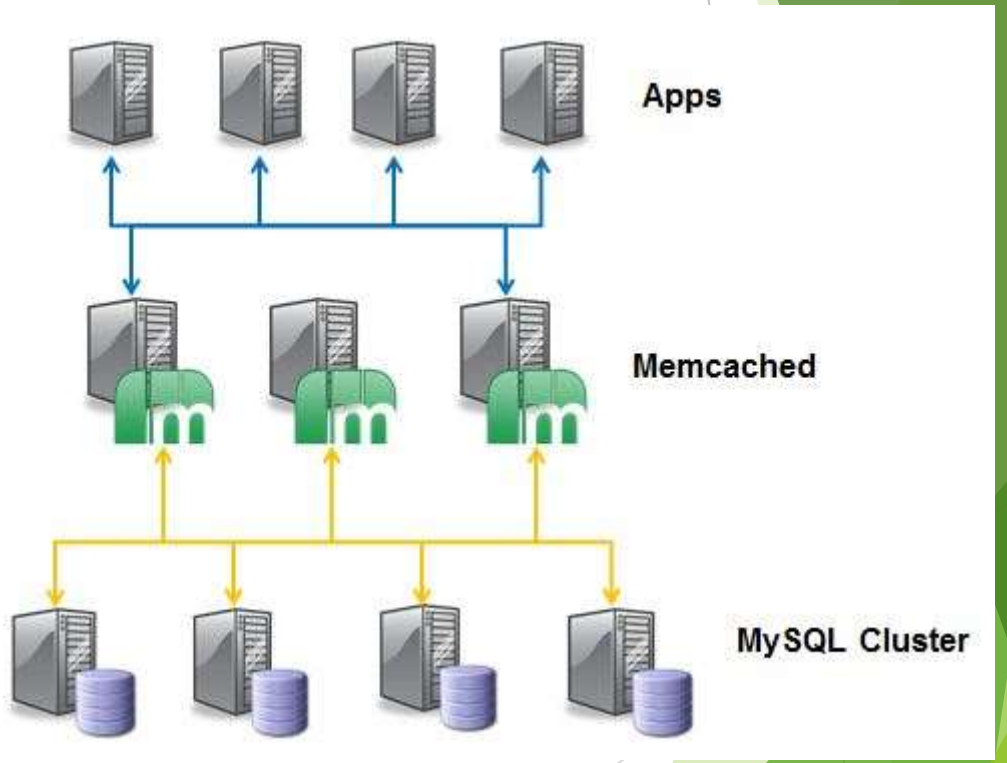
Relational databases have become such an embedded part of our computing culture that it's easy to take them for granted. It's therefore useful to revisit the benefits they provide.



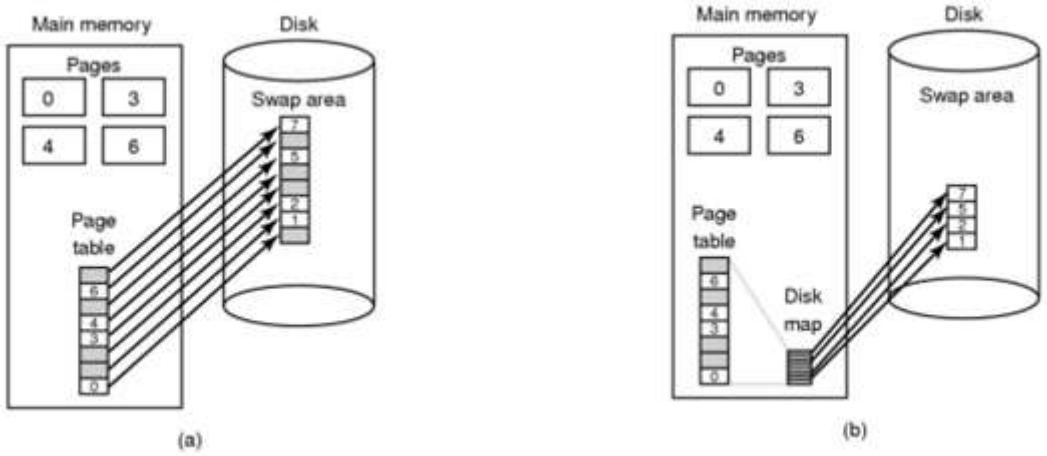
# Getting at Persistent Data

Probably the most obvious value of a database is keeping large amounts of persistent data. Most computer architectures have the notion of two areas of memory: a fast volatile "**main memory**" and a larger but slower "**backing store**."

Main memory is both limited in space and loses all data when you lose power or something bad happens to the operating system. Therefore, to keep data around, write it to a backing store, commonly seen a disk (although these days that disk can be persistent memory).



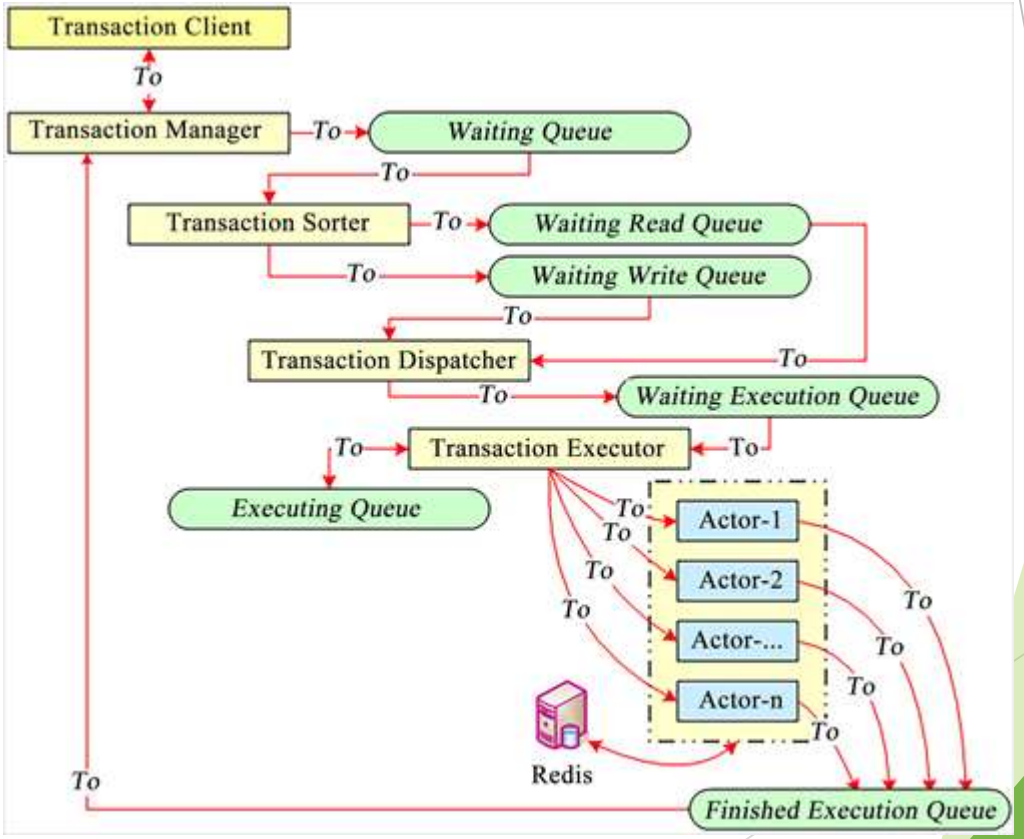
The **backing store** can be organized in all sorts of ways. For many productivity applications (such as word processors), it's a file in the file system of the operating system. For most enterprise applications, however, the backing store is a database. The database allows more flexibility than a file system in storing large amounts of data in a way that allows an application program to get at small bits of that information quickly and easily.



## Concurrency

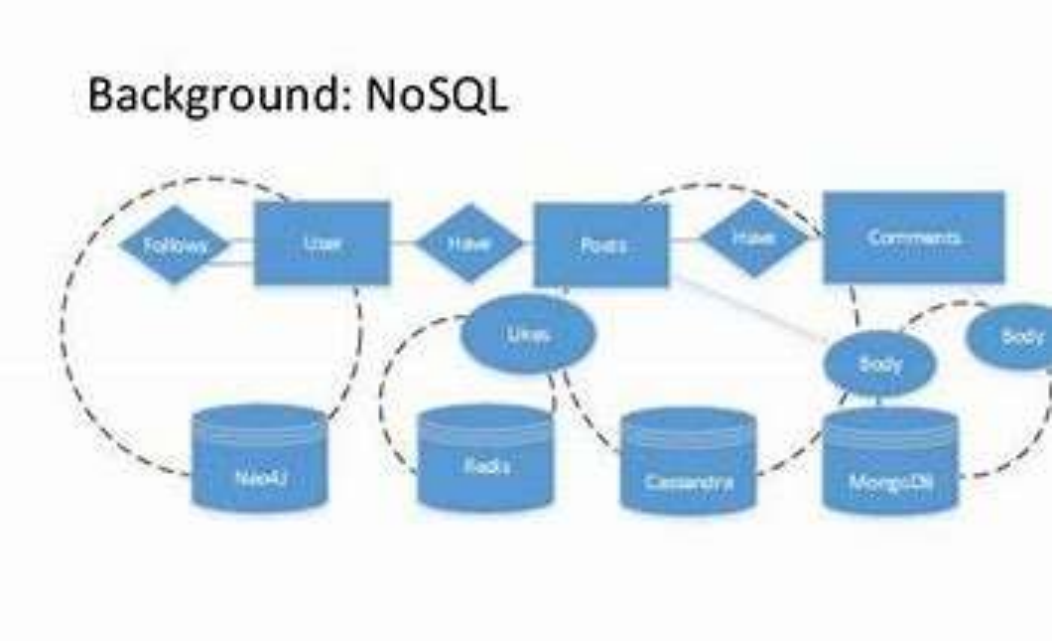
Enterprise applications tend to have many people looking at the same body of data at once, possibly modifying that data. Most of the time they are working on different areas of that data, but occasionally they operate on the **same bit of data**.

**Example :** Have to worry about coordinating these interactions to avoid such things as double booking of hotel rooms.



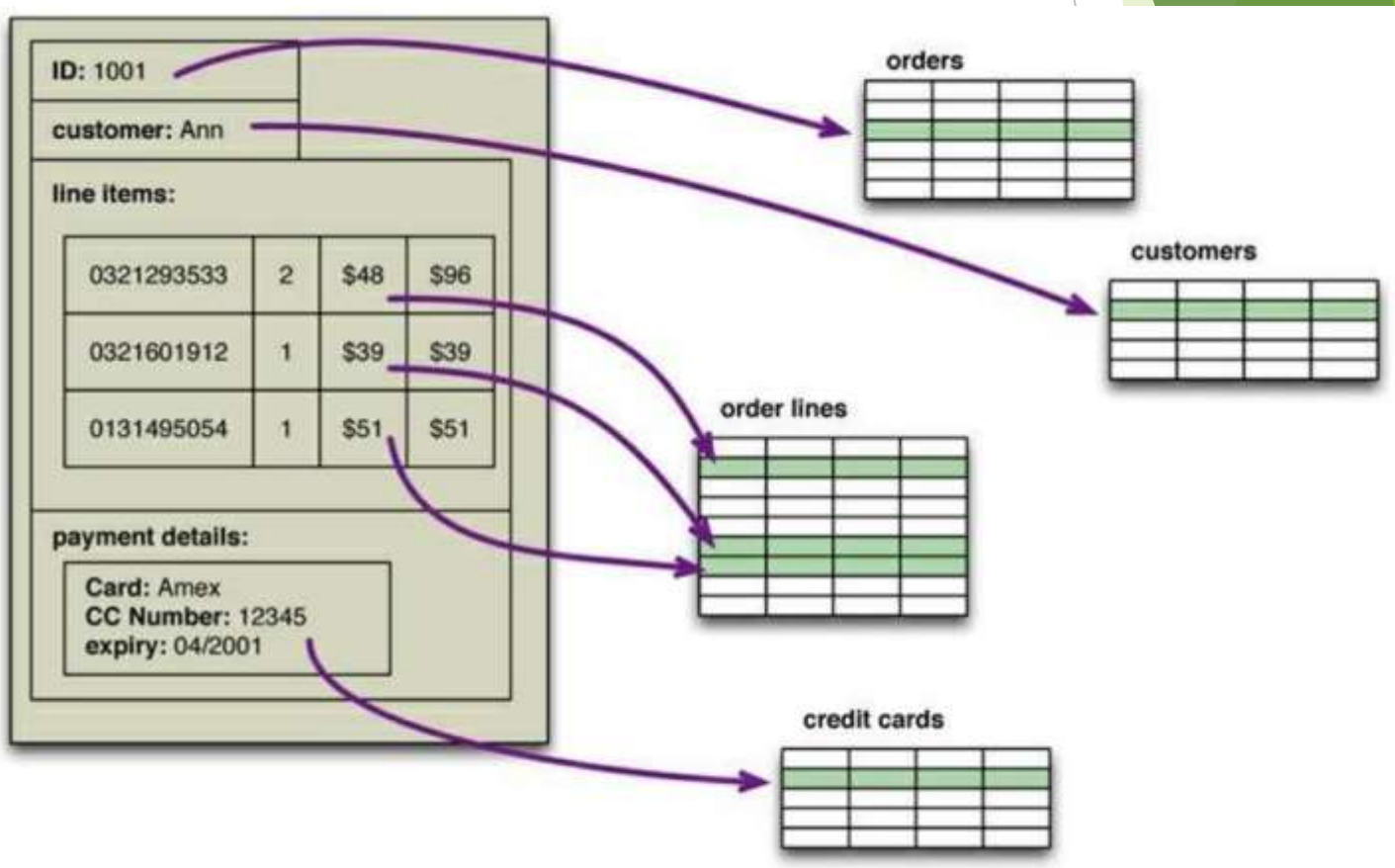
## Integration

A common way to do this is **shared database integration** ,where multiple applications store their data in a single database. Using a single database allows all the applications to use each others' data easily, while the database's concurrency control handles multiple applications in the same way as it handles multiple users in a single application.



## Impedance Mismatch

Impedance mismatch is a term used in computer science to describe the problem that arises when two systems or components that are supposed to work together have different data models, structures, or interfaces that make communication difficult or inefficient.

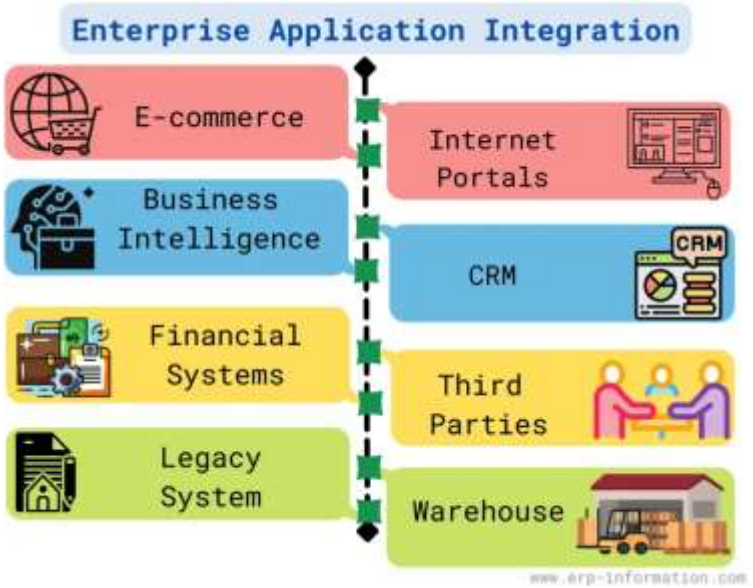


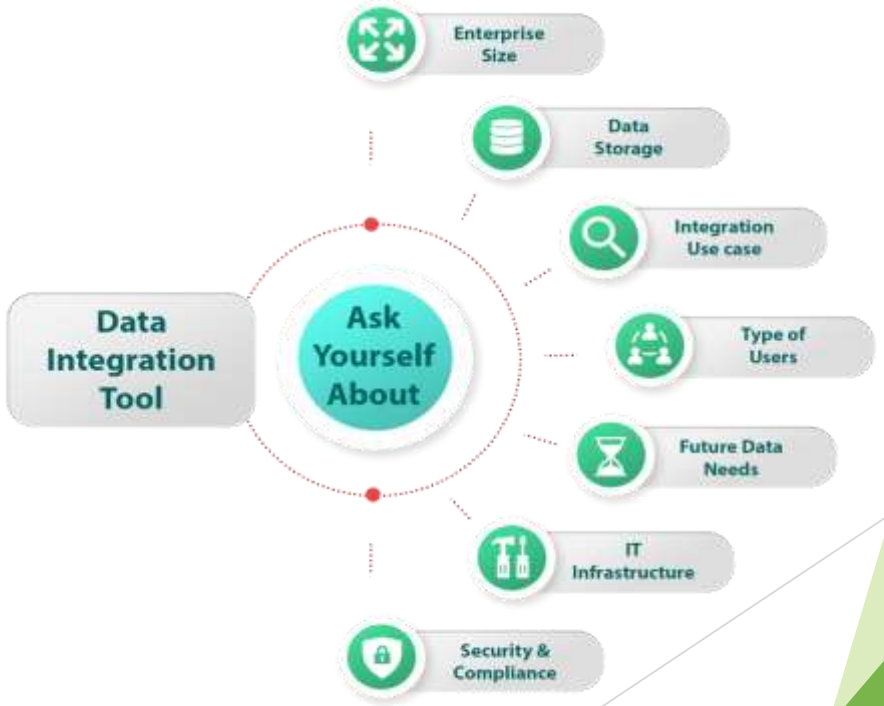
## Application integration

An application integration creates connectors between two or more applications so they can work with one another.

## Data integration

Data integration is the process of taking data from different sources and formats and combining it into a single data set.



# Common use cases

**Application integration is typically used to do the following:**

1. Collect data from Internet of Things (IoT) devices that will be stored and used for analytics.
2. Sync legacy, on-premises ERP systems to CRMs.
3. Build automations between applications for more efficient workflows.

**Data integration is typically used to do the following:**

1. Migrate data into a data warehouse.
2. Integrate and consolidate customer data from disparate sources into a single view.
3. Assist with a transition to a multicloud or hybrid cloud environment.

# Clusters

Clusters in SQL are used to store data that is from different tables in the same physical data blocks. They are used if records from those tables are frequently queried together. By storing same data blocks, the number of database block reads needed to full fill such queries decreases which improves performance.

```
CREATE CLUSTER <Cluster Name>
(<Column> <Data Type> [, <Column> <Data Type> ] . . . . . ) [<Other Options >]
```

```
create table branch_master
( "branch_code" varchar(10) PRIMARY KEY,
"branch_name" varchar(15) );
```

**Advantages of clusters :**

1. Disk I/O is reduced.
2. Access time improves for joins of clustered tables.
3. As all rows in clustered tables use the same columns as common primary key, this yields storage benefit.

**Disadvantages of clusters :**

1. Reduces performance of INSERT statements as compared to storing the table separately with its own index.
2. Columns that are often updated are not good candidates for cluster key.

# The emergence of NoSQL Databases

In the presence of clustering and the problems when clusters work with relational databases, coupled with the impendance mismatch problem discussed earlier, a number of database approaches came forward. These are collectively called "NoSQL", though this is hardly an apt term.

These databases vary a lot in their approach:

**column-oriented databases**

like BigTable, Cassandra and HBase focus on column families as the primary unit of storage, as opposed to table rows. In effect each row contains sets of column values organized in "families". This provides a richer structure to the normal relational database approach.

**document-oriented databases**

like MongoDB and CouchDB focus on complex aggregate units called *documents* as their primary unit of storage. We can loosely think of a document as a JSON structure with all the information contained therein.

**graph databases**

like FlockDB and Neo4j focus on relations between objects, and they are particularly effective at encoding the complex interactions in social networks.

**key-value stores**

like Riak and Redis offer very simple storage of values based on their keys. As a result they can be extremely efficient and fast. They are often used as in-memory cache stores.

What they all have incommon is that they do not follow the relational model, that they are schema-less, and that they are designed to perform well in clustered settings.