

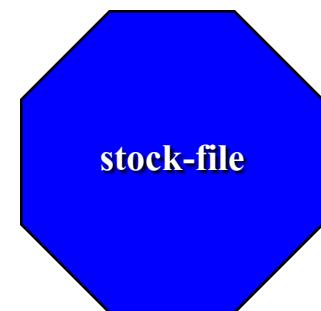
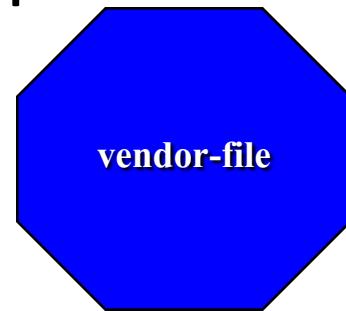
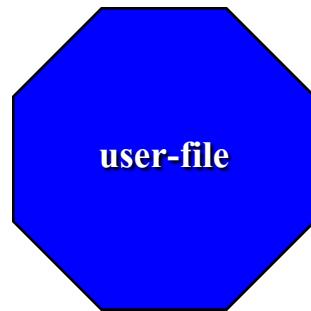
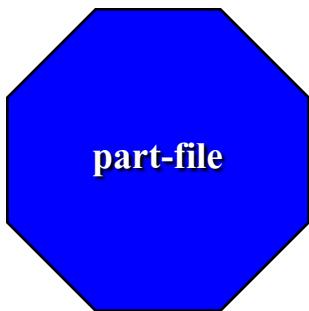
# Content

- Introduction to DBMS
- Basic Database Terminology
- Types of DBMS
  - Relational
  - Object Relational
  - NoSQL Databases
- Introduction to MySQL, MySQL Clients (Monitor, Shell, Workbench)

# Conventional Systems

First .....

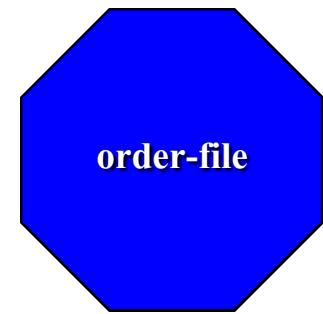
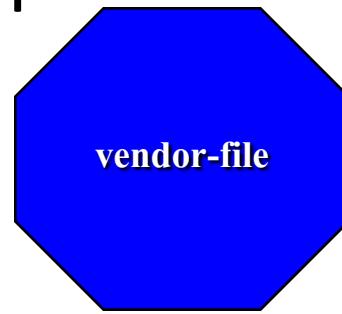
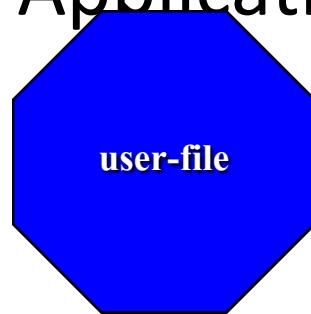
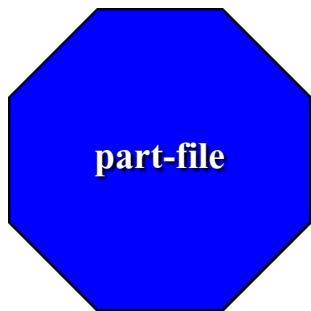
An Inventory Control Application



# Conventional Systems

Then .....

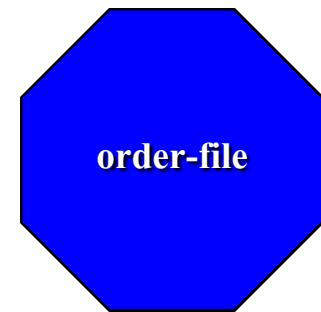
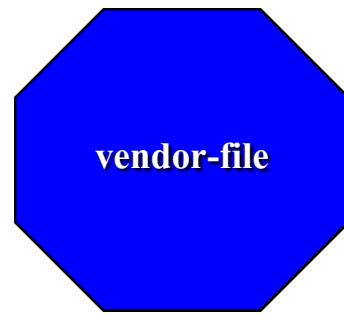
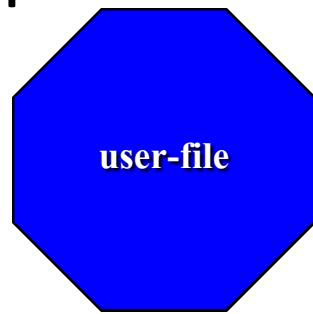
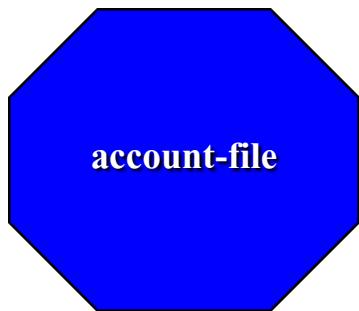
A Purchasing Application



# Conventional Systems

and then .....

A Finance Application



# Conventional Systems

and Now .....

The Information Seeker!



3 crores!

2.7 crores!

2.3 crores!

3.4 crores!

# Problems

- Data Redundancy
- No consistency
- No integrity
  - maintaining and assuring the accuracy
- Concurrency related problems
  - Note: OS concurrency never handled database concurrency implementations
- Security – Not everyone should be allowed to access all the data at all the time

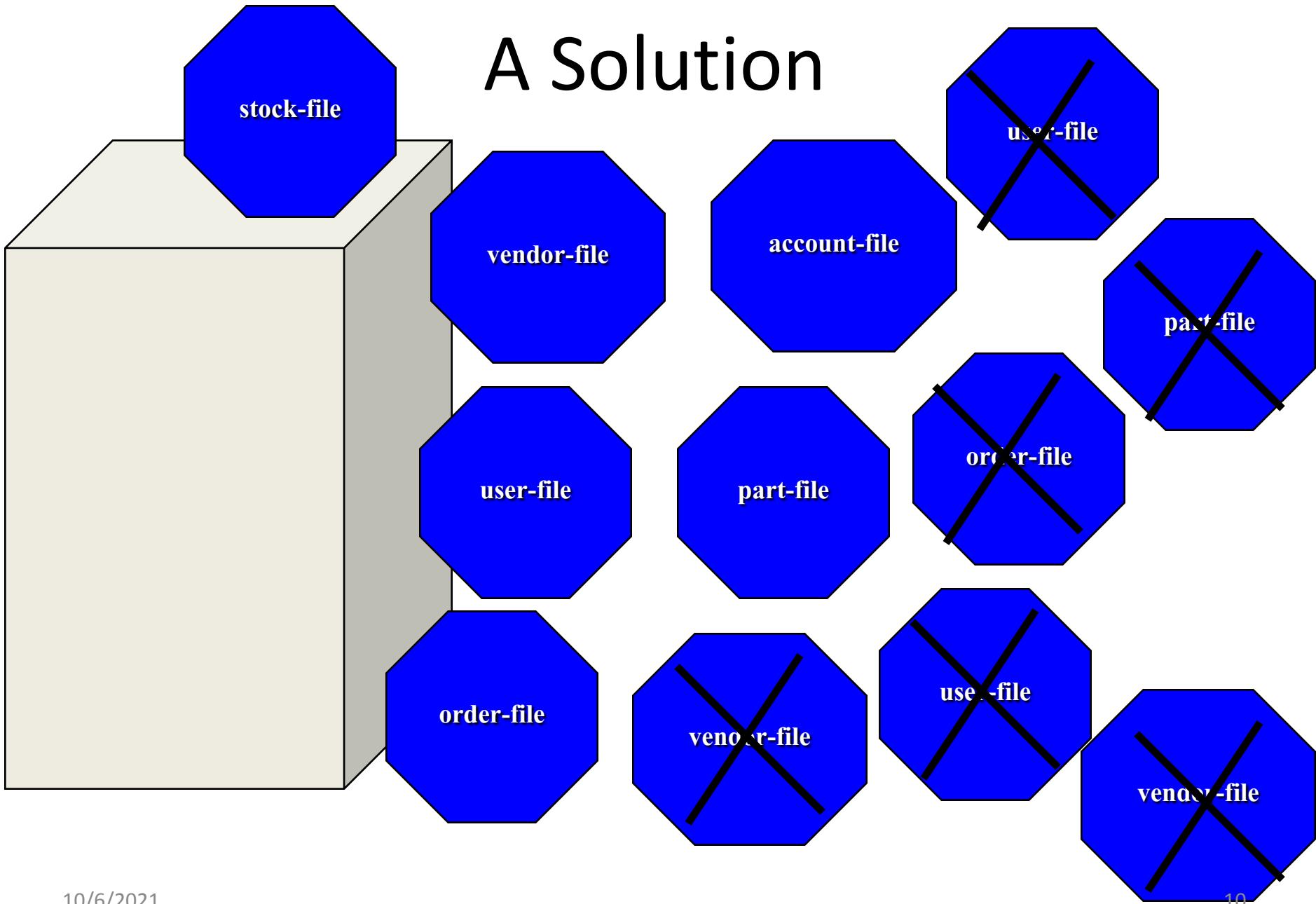
# The Causes?

- No sharing
- Data isolation
- Diffused responsibilities
- Poor coordination
- Disorganized developments
- Data redundancy
- Weak integrity

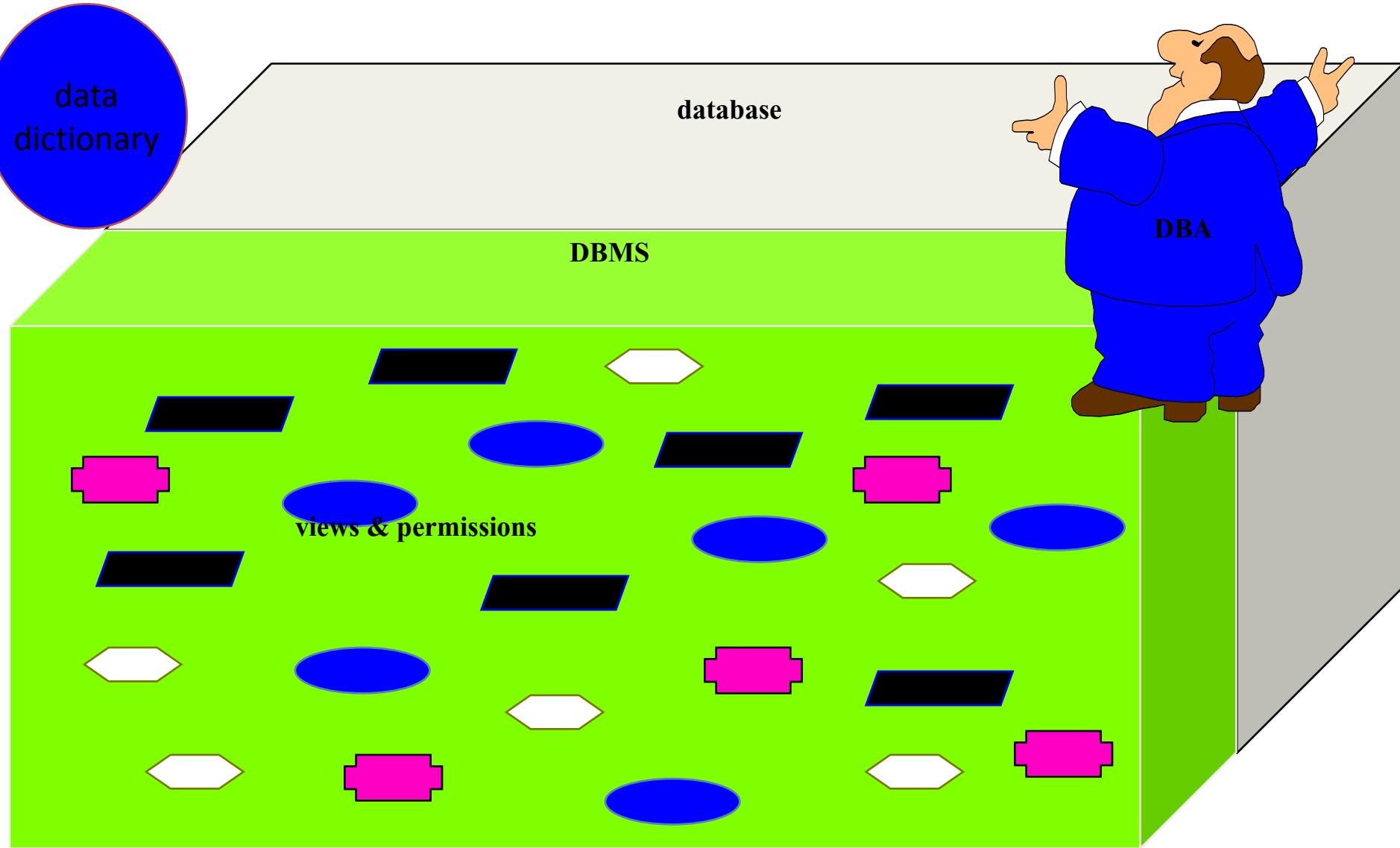
# Problems with File systems

- Problems with file processing systems
  - Limited data sharing
  - Poor enforcement of standards
  - Inconsistent data
  - Inflexibility – everything dependent on programs
  - Even simple tasks required extensive programming
  - Security feature practically nil
  - Complex system administration
  - Excessive program maintenance

# A Solution



# A Solution



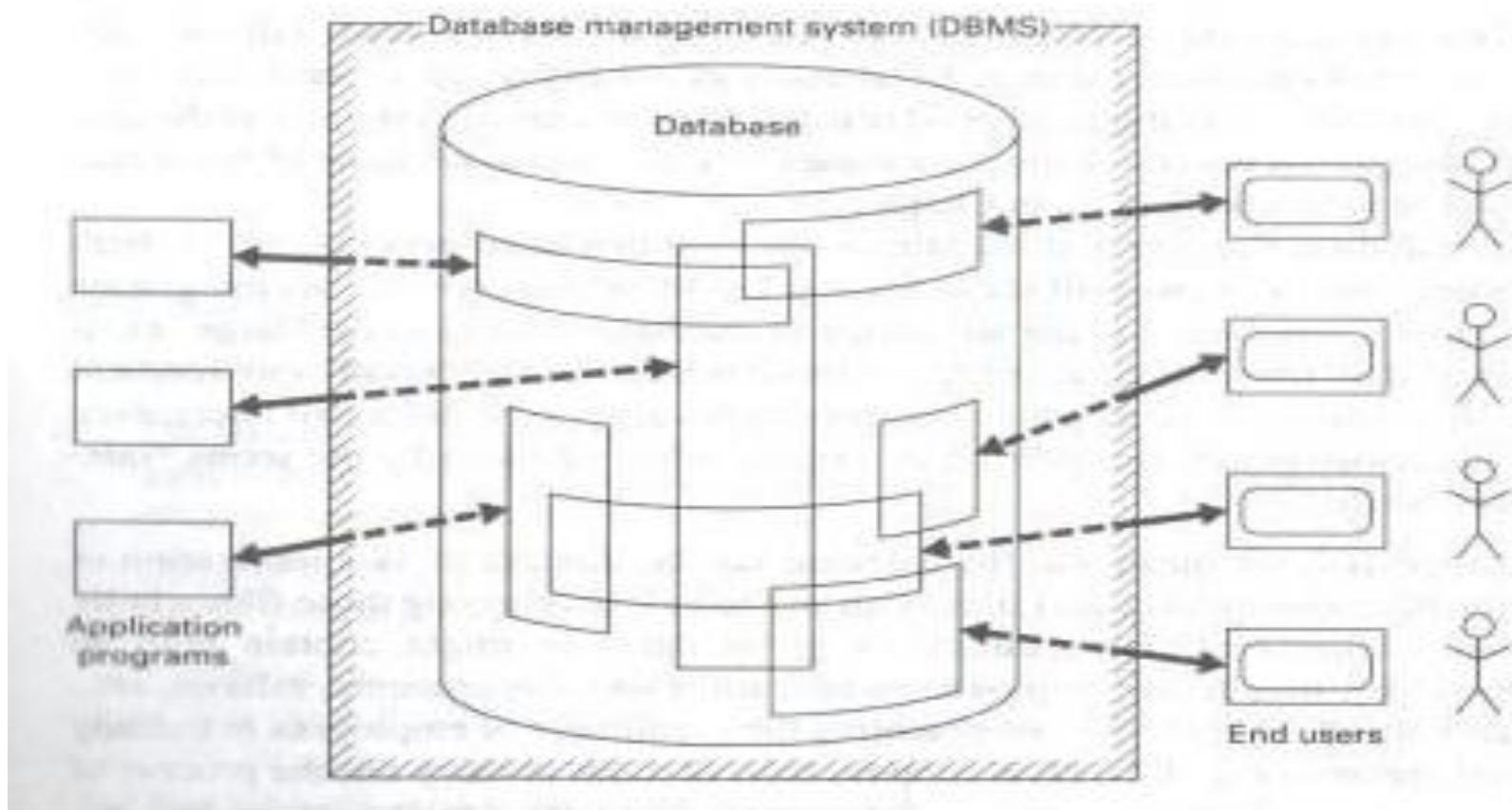
# *Data, Information, Database*

- Data –raw facts
- Information- Result of processing raw data to reveal meaning
- So data is building blocks of information
- Information produced by processing raw data
- Accurate, timely, relevant information is the key to good decision making
- A database is a collection of information that is organized so that it can easily be accessed, managed, and updated.
- Database- Shared, integrated computer structure that stores a collection of end-user data and meta data

# Introduction to DBMS

- A database management system (DBMS) is system software for creating and managing databases.
- Data Base Management system- collection of programs responsible for managing the structure and control access to data
- The DBMS provides users and programmers with a systematic way to create, retrieve, update and manage data.

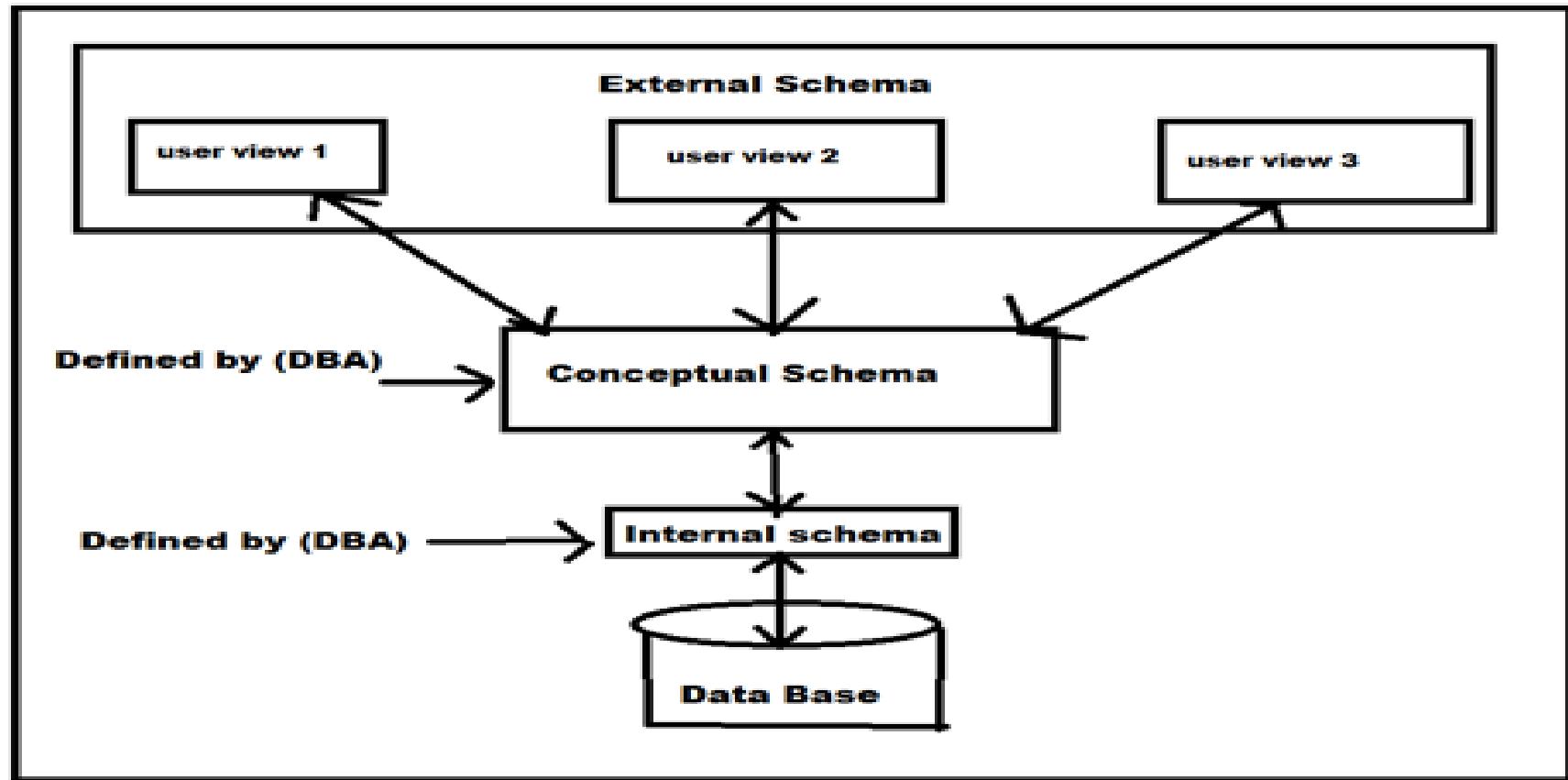
# Introduction to DBMS



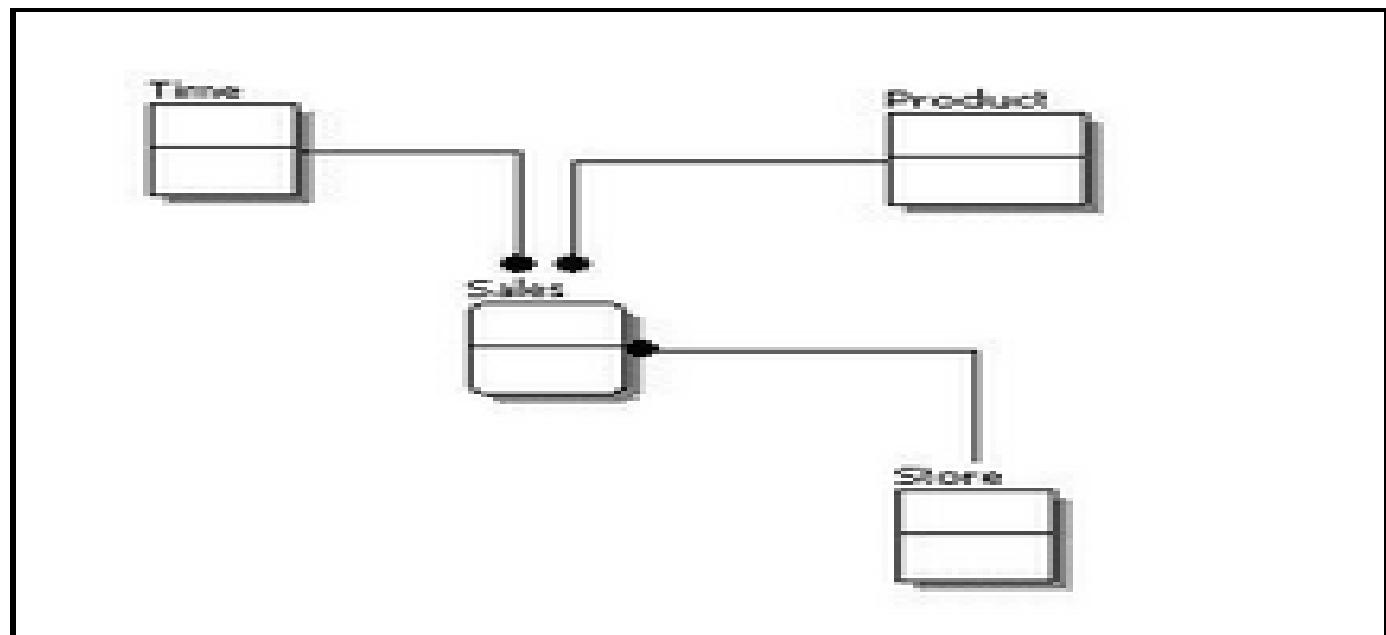
# Architecture for a DBMS

- External view-Content of the database as seen by a particular external user defined by external schema
- Conceptual view- Abstract view of the physical level defined by conceptual schema
- Internal view- Low level representation of the entire database defined by internal schema

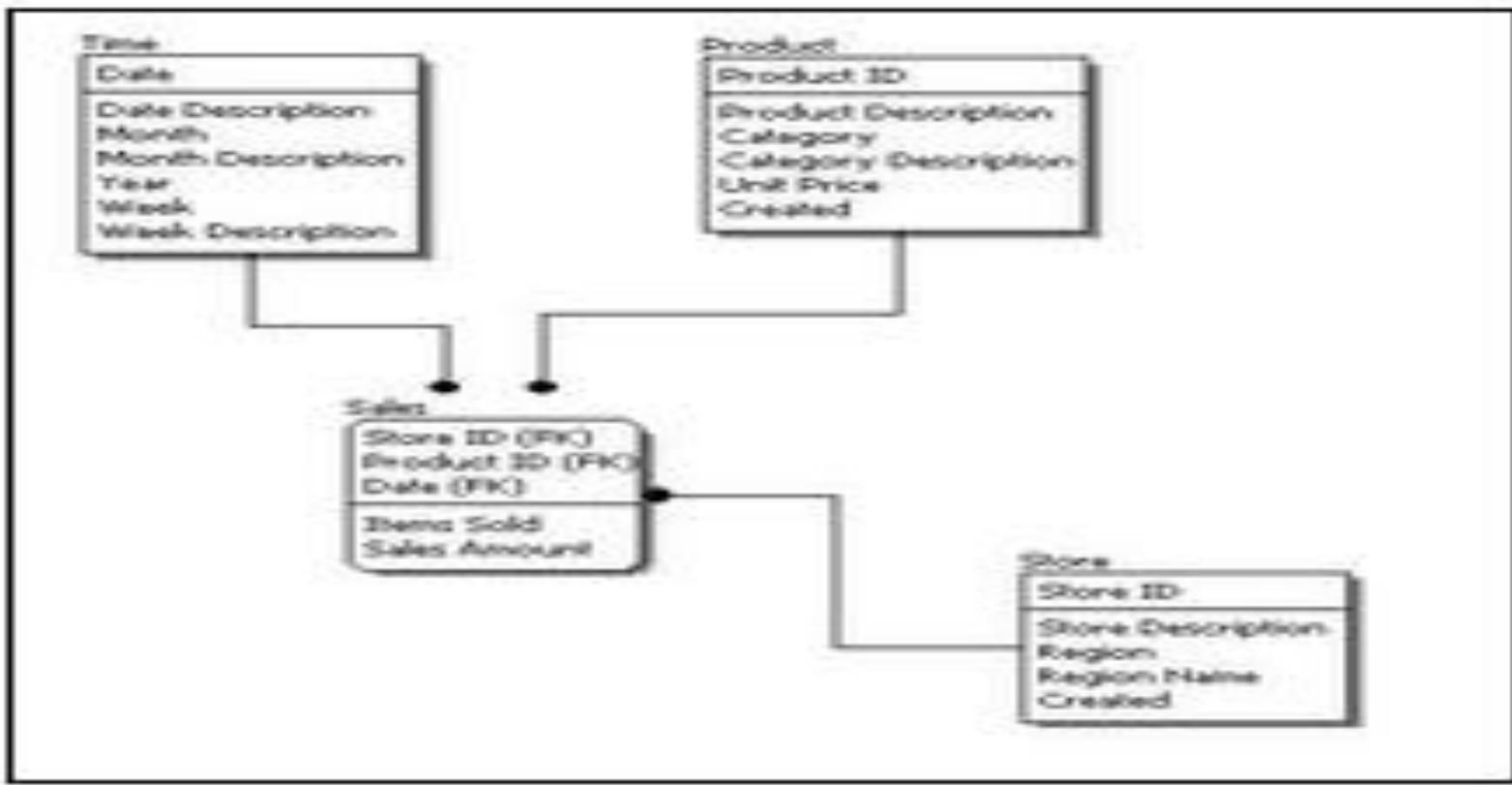
# Architecture for a DBMS



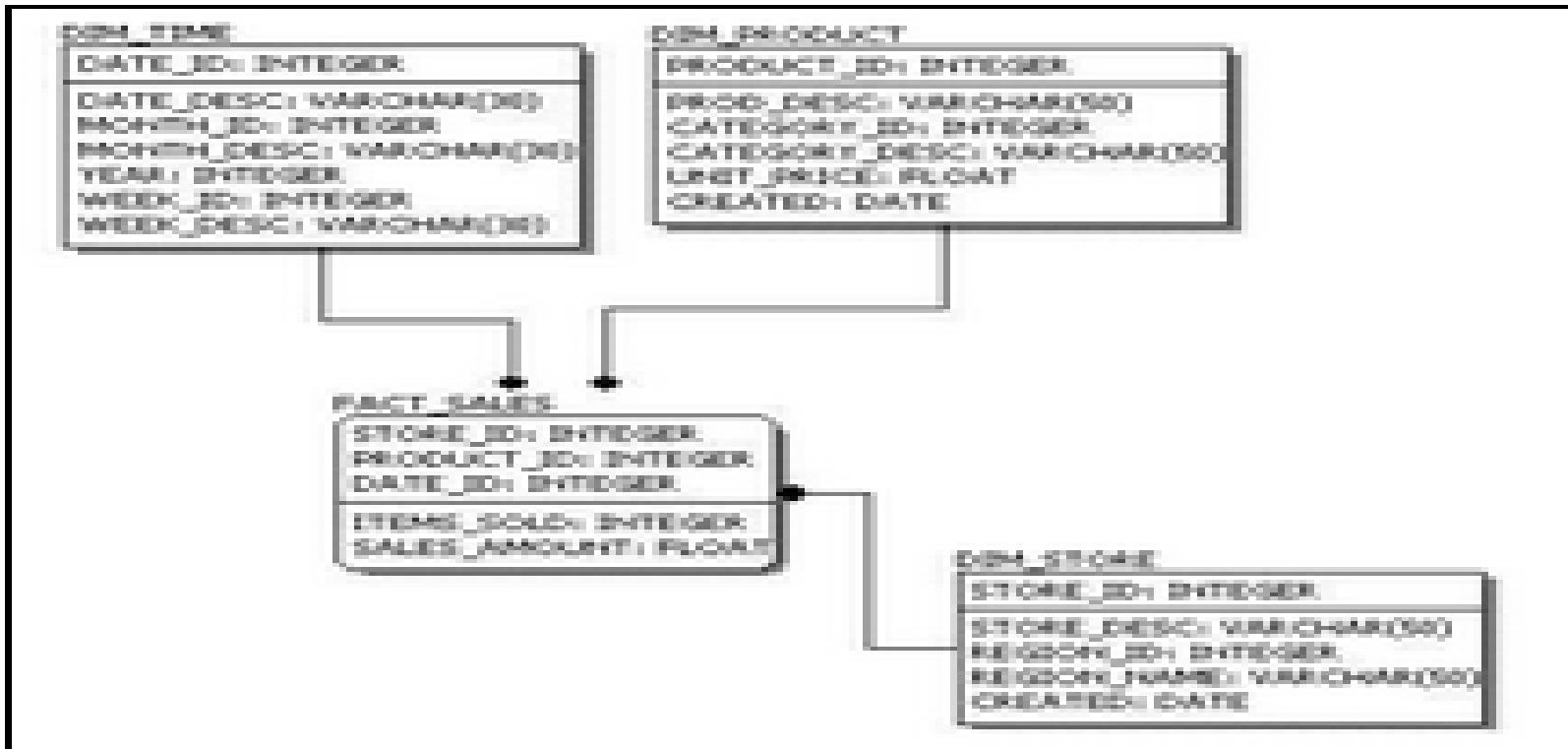
# Conceptual Model Design



# Logical Model Design



# Physical Model Design



# Data Models

- Three Components

1. Structures

- rows and columns?
- nodes and edges?
- key-value pairs?
- a sequence of bytes?

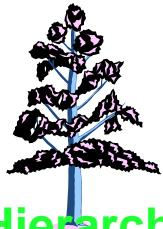
2. Constraints

- all rows must have the same number of columns
- all values in one column must have the same type
- a child cannot have two parents

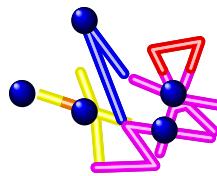
3. Operations

- find the value of key x
- find the rows where column “lastname” is “Jordan”
- get the next N bytes

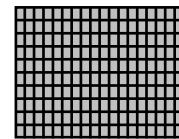
# Types of databases



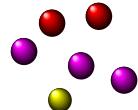
Hierarchical  
databases



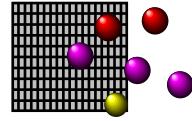
Network  
databases



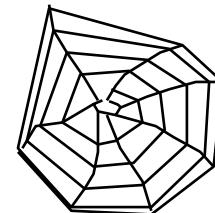
Relational  
databases



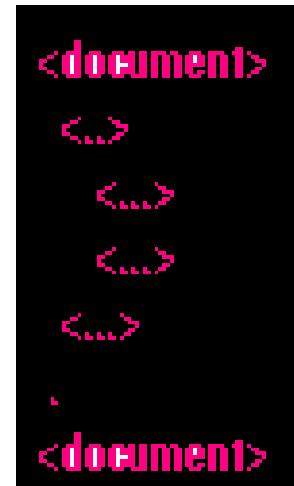
Object  
databases



Object-relational  
databases



Enterprise  
databases



Document-  
oriented  
databases

# Relational Model

**table**

course

module

student

**attributes**

faculty

book

test

assign

perf

attend

fee

session

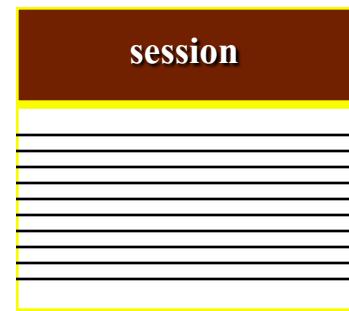
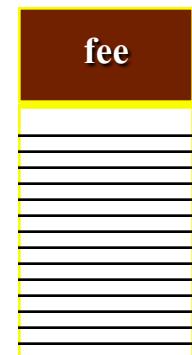
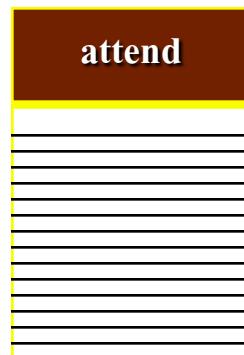
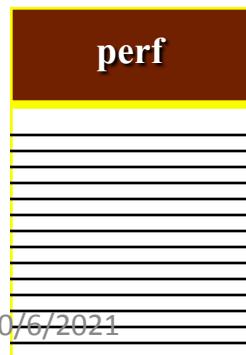
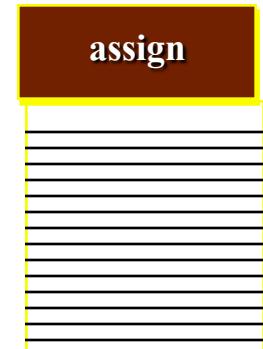
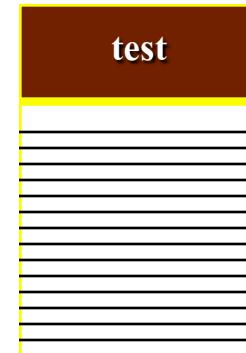
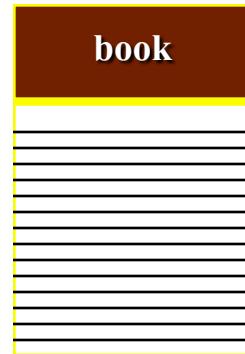
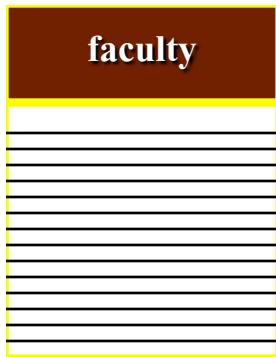
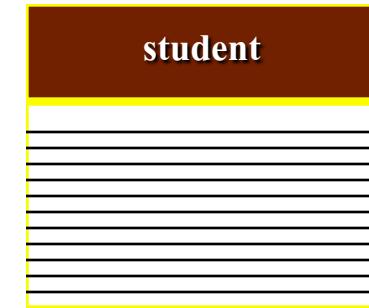
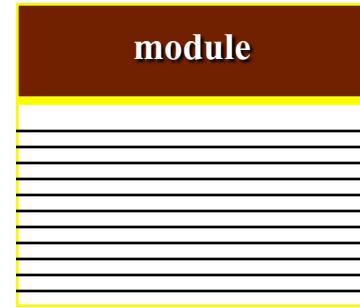
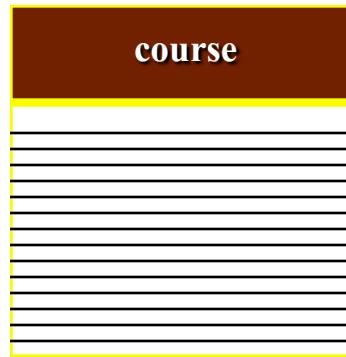
# Relational model

- A DBMS is said to be a Relational DBMS or RDBMS if the database relationships are treated in the form of a table.
- Three keys on relational DBMS
  - Relation
  - Domain
  - Attributes.
- A number of RDBMSs are available, some popular examples are Oracle, Sybase, Ingress, Informix, Microsoft SQL Server, and Microsoft Access.

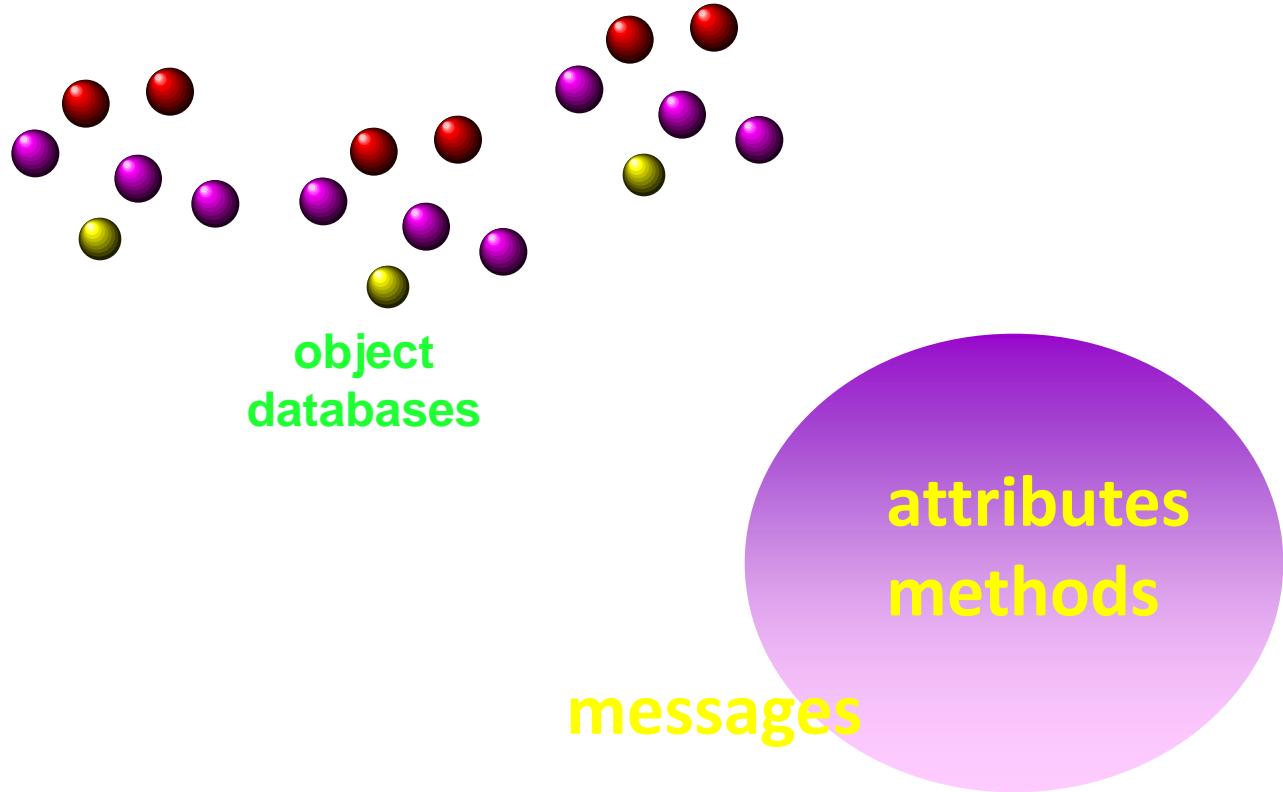
# Relational model (contd)

- Properties of Relational Tables:
  - Values Are Atomic
  - Each Row is Unique
  - Column Values Are of the Same Kind
  - The Sequence of Columns is Insignificant
  - The Sequence of Rows is Insignificant
  - Each Column Has a Unique Name

# Relational Database



# Object-Oriented Model



# Object-Oriented Model

- New user defined data types.
- Complex data types
- Object references and methods
- New capabilities like encapsulation, inheritance etc. for databases.
- Intuitive and Natural Model

# What is Object Oriented Database? (OODB)

- A database system that incorporates all the important object-oriented concepts
- Some additional features
- Unique Object identifiers
- Persistent object handling
- Designer can specify the structure of objects and their behavior (methods)
- Better interaction with object-oriented languages such as Java and C++
- Definition of complex and user-defined types
- Encapsulation of operations and user-defined methods

- Queries look very similar in SQL and OQL, sometimes they are the same
- In fact, the results they give are very different Query returns
- Foundation for several OO database management systems – ORACLE8, DB2, etc

OQL	SQL
Object	Tuple
Collection of objects	Table

# Benefits of OODBMS

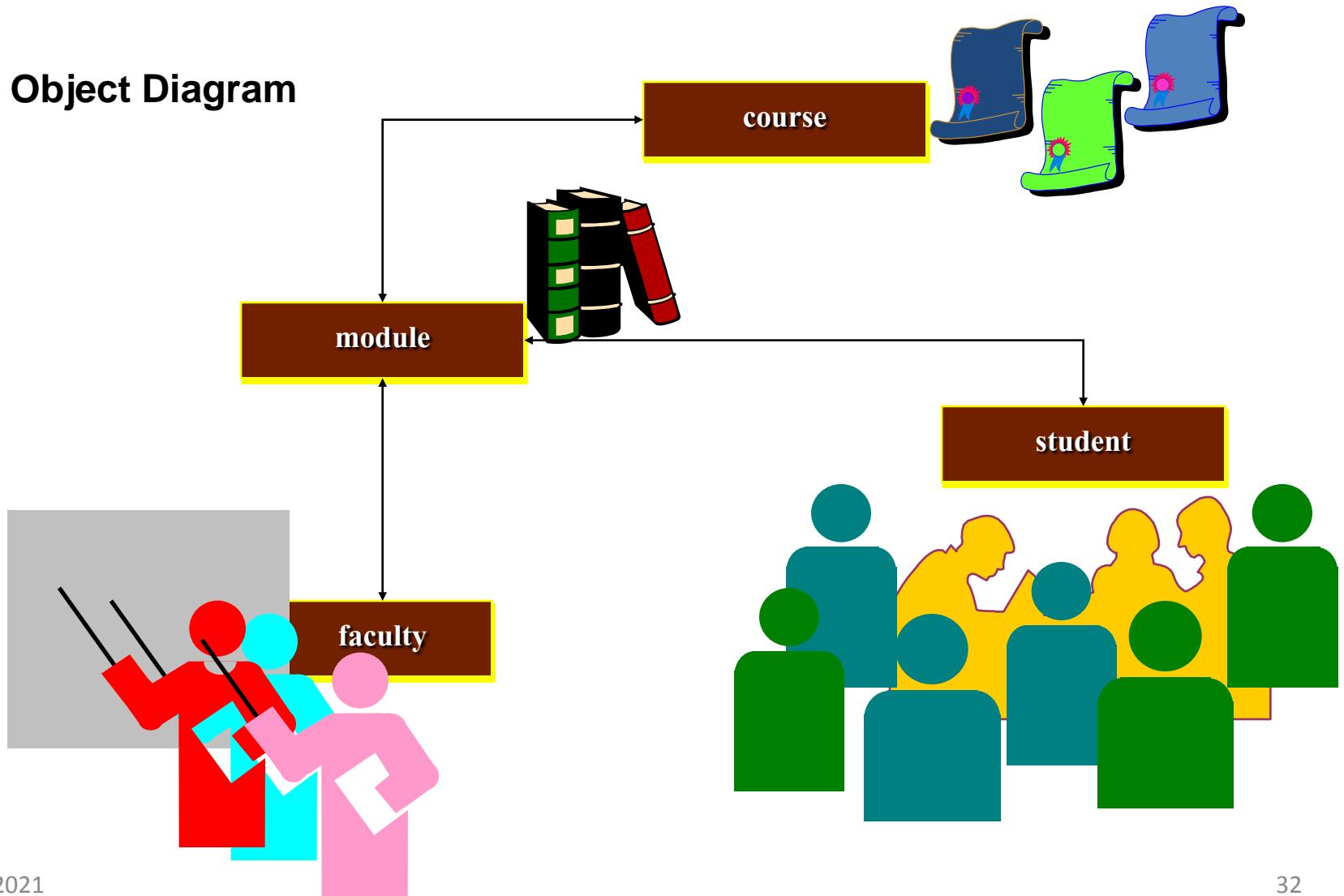
- Object database is a good choice for three factors: business need, high performance, and complex data.
- With ODBMS, lesser code is required compared to RDBMS.
  - If using Java or C++ -- no need to translate into a database sub-language such as SQL, ODBC, or JDBC.
  - The data structure that you can imagine in Java or C++ can be stored directly without translation in an ODBMS.
- ODBMS give better performance than an RDBMS.
  - The data is read off the disk, it is already in the format that Java or C++ uses (OO). No translation is needed.

# Shortcomings OODBMS

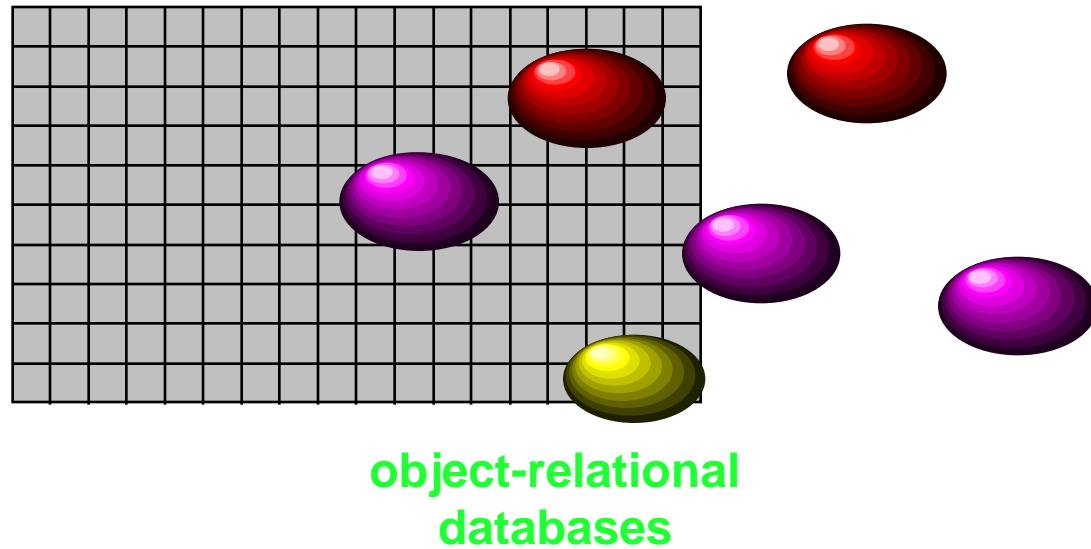
- Object databases are not as popular as RDBMS. It is difficult to find object DB developers.
- Not many programming language support object databases.
- RDBMS have SQL as a standard query language. Object databases do not have a standard.
- Object databases are difficult to learn for non-programmers.

# Object Model

Object Diagram



# Object-Relational Model



# Object-Relational Model

- Add ‘object oriented-ness’ to tables
  - Combination of
    - OO features - Complex objects, Functions, Inheritance, Overloading
- And
- Relations features - Tables, Views, Transactions, Recovery, Indexing, Optimization, SQL queries
  - Data is still stored in tables
  - SQL3 ('object-oriented' SQL) is the language for data definition, manipulation, and query.

# Object-Relational Model

- Extend the relational data model by including object orientation and constructs to deal with added data types.
- Allow attributes of tuples to have complex types, including non-atomic values such as nested relations.
- Preserve relational foundations, in particular the declarative access to data, while extending modeling power.
- Upward compatibility with existing relational languages.
- PostgreSQL is the most popular pure ORDBMS.
- Some popular databases including Microsoft SQL Server, Oracle, and IBM DB2 also support objects and can be considered as ORDBMS.

# Stonebraker's Application Matrix

	No Query	Query
Complex Data	OODBMS	ORDBMS
Simple Data	File System	RDBMS

Stonebraker's view: Most applications will move to the upper right.

# Current and Emerging Trends

- Analytics
  - Data Warehousing
  - Data Mining
  - Big Data
- Types
  - Document oriented databases
  - Multimedia databases
  - Distributed databases
  - Mobile and Embedded databases

# Conclusion

- Data: Known facts that can be recorded and have implicit meaning
- Database: Collection of interrelated data
- DBMS: A computerized data/record keeping system for managing data

# Introduction to MySQL

- Open Source SQL database management system
- Developed, distributed, and supported by Oracle Corporation.
- MySQL Database Server is very fast, reliable, scalable, and easy to use.

# Introduction to MySQL

MySQL Support by Platform		
Operating System	Architecture	Version
Oracle Linux / Red Hat / CentOS		8.0    5.7
Oracle Linux 8 / Red Hat Enterprise Linux 8 / CentOS 8	x86_64, ARM 64	▪
Oracle Linux 7 / Red Hat Enterprise Linux 7 / CentOS 7	ARM 64	▪
Oracle Linux 7 / Red Hat Enterprise Linux 7 / CentOS 7	x86_64	▪    ▪
Oracle Linux 6 / Red Hat Enterprise Linux 6 / CentOS 6	x86_32, x86_64	▪    ▪
Oracle Solaris		
Solaris 11 (Update 4+)	SPARC_64	▪    ▪
Canonical		
Ubuntu 21.04	x86_64	▪
Ubuntu 20.04 LTS	x86_64	▪
Ubuntu 18.04 LTS	x86_32, x86_64	▪    ▪
SUSE		
SUSE Enterprise Linux 15 / OpenSUSE 15 (15.2)	x86_64	▪
SUSE Enterprise Linux 12 (12.5+)	x86_64	▪    ▪
Debian		
Debian GNU/Linux 11	x86_64	▪

# Introduction to MySQL

MySQL 8.0.25			
Platform	Architecture	Binary	Source
Microsoft Windows Server			
Microsoft Windows 2019 Server	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Microsoft Windows 2016 Server	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Microsoft Windows 2012 Server R2	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Microsoft Windows			
Microsoft Windows 10	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Apple			
macOS 11	x86_64, ARM_64	<a href="#">Download</a>	<a href="#">Source</a>
macOS 10.15	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Various Linux			
Generic Linux (tar format)	x86_32, x86_64, glibc 2.12, libstdc++ 4.4	<a href="#">Download</a>	<a href="#">Source</a>
Yum Repo			
APT Repo			
SUSE Repo			

# Introduction to MySQL

MySQL 8.0.25			
Platform	Architecture	Binary	Source
Microsoft Windows Server			
Microsoft Windows 2019 Server	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Microsoft Windows 2016 Server	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Microsoft Windows 2012 Server R2	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Microsoft Windows			
Microsoft Windows 10	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Apple			
macOS 11	x86_64, ARM_64	<a href="#">Download</a>	<a href="#">Source</a>
macOS 10.15	x86_64	<a href="#">Download</a>	<a href="#">Source</a>
Various Linux			
Generic Linux (tar format)	x86_32, x86_64, glibc 2.12, libstdc++ 4.4	<a href="#">Download</a>	<a href="#">Source</a>
Yum Repo			
APT Repo			
SUSE Repo			

# MySQL Client – Command Line

```
(base) surabhi@surabhi-seng:~$ sudo mysql
[sudo] password for surabhi:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 14
Server version: 8.0.26-0ubuntu0.20.04.2 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement

mysql> show databases;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| sys            |
+-----+
4 rows in set (0.00 sec)

mysql> █
```

# MySQL Workbench

The screenshot shows the MySQL Workbench application window. The title bar reads "MySQL Workbench" and "dev\_server". The menu bar includes File, Edit, View, Query, Database, Server, Tools, Scripting, and Help. The toolbar has icons for SQL, Scripts, Databases, Servers, Tools, and Help.

The left sidebar contains the "Navigator" section with "MANAGEMENT" and "INSTANCE" categories, and the "MySQL ENTERPRISE" section with Audit Inspector, Online Backup, and Backup Recovery. The "SCHEMAS" section lists databases: information\_schema, mysql, performance\_schema, sakila, and test. Under the test database, there are tables: city and country. The "Information" section shows the "Table: city" with columns: ID, Name, CountryCode, District, and Population.

The main content area displays the "Administration - Server Status" tab. It shows the connection details for "dev\_server": Host MFRANK-US, Socket MySQL, Port 3306, Version 5.6.12-enterprise-commercial-advanced MySQL Enterprise Server - Advanced Edition (Commercial), and Compiled For Win64 (x86\_64). Below this, the "Available Server Features" section lists various server settings with their current status (e.g., On or Off).

The right side of the screen features a dashboard with several performance metrics: Server Status (Running), CPU usage (25%), Connections (4), Traffic (21.80 KB/s), Key Efficiency (76.6%), Queries per Second (983), InnoDB Buffer Usage (3.7%), InnoDB Reads per Second (0), and InnoDB Writes per Second (989). The overall interface is clean and modern, designed for managing MySQL databases.

# MySQL Workbench

## Design

- MySQL Workbench enables a DBA, developer, or data architect to visually design, model, generate, and manage databases.
- It includes everything a data modeler needs for creating complex ER models, forward and reverse engineering,



## Diagram



Customer Data

## Indexes

## Indexes

## Customer related data

## Indexes

## Indexes

## Indexes

## Indexes

## Indexes

## Indexes

## Inventory

## Indexes

## Indexes

## Triggers

## AFT UPDATE upd\_film

## AFT INSERT ins\_film

## AFT DELETE del\_film

## Indexes

## Indexes

## Indexes

## Movie database

## VIEWS

## film\_list

## nicer\_but\_slower\_film\_list

## actor\_info

## sales\_by\_store

## sales\_by\_film\_category

## staff\_list

## customer\_list

## Film

## film\_in\_stock

## film\_not\_in\_stock

## Zhoules

## Catalog Tree

- sakila
  - Tables
    - actor
    - address
    - category
    - city
    - country
    - customer
    - film
    - film\_actor
    - film\_category
    - film\_text
    - inventory

## Catalog Layers User Types

## Properties Editor

Name	Value
color	#499E21
expanded	True
height	258
indicesExpanded	False
left	37
locked	False
manualSizing	True
name	staff
summarizeDisplay	-1
top	61
triggersExpanded	False
width	120

## Description

## Properties

## H

## Reverse Engineer Database

Connection Options  
Connect to DBMS  
Select Schemas  
Retrieve Objects  
**Select Objects**  
Reverse Engineer

Results

### Select Objects to Reverse Engineer



Import MySQL Table Objects

16 Total Objects, 9 Selected

[Hide Filter](#)

sakila.film\_category  
sakila.film\_text  
sakila.inventory  
sakila.language  
sakila.new\_actors  
sakila.payment  
sakila.rental  
sakila.staff  
sakila.store

>  
<  
=>  
<<  
+

sakila.actor  
sakila.address  
sakila.category  
sakila.country  
sakila.customer  
sakila.film  
sakila.film\_actor

Use the + button to exclude objects matching wildcards such as \* and \_



Import MySQL View Objects

[Show Filter](#)

7 Total Objects, 7 Selected

Place imported objects on a diagram

[Back](#)

[Execute >](#)

[Cancel](#)

# MySQL Workbench

## Develop

- MySQL Workbench delivers visual tools for creating, executing, and optimizing SQL queries.
- The SQL Editor provides color syntax highlighting, auto-complete, reuse of SQL snippets, and execution history of SQL.
- The Database Connections Panel enables developers to easily manage standard database connections
- The Object Browser provides instant access to database schema and objects.



## Navigator

## MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

## INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

## MySQL ENTERPRISE

- Audit Inspector
- Online Backup
- Backup Recovery

## SCHEMAS

Filter objects

▶	customer
▼	film
▼	Columns
▶	film_id
▶	title
▶	description
▶	release_year
▶	language_id
▶	original_language_id
▶	rental_duration
▶	rental_rate
▶	length

## Information

## Table: film

## Columns:

<u>film_id</u>	smallint(5) U
<u>title</u>	varchar(255) U
description	text
release_year	year(4)
<u>language_id</u>	tinyint(3) UN
<u>original_language_id</u>	tinyint(3) UN
rental_duration	tinyint(3) UN
rental_rate	decimal(4,2)
length	smallint(5) U

## Object Info Session

## Query 1

```

1 •   SELECT `actor`.`actor_id`,
2       `actor`.`first_name`,
3       `actor`.`last_name`,
4       `actor`.`last_update`
5   FROM `sakila`.`actor`;
6
7 •   SELECT `film`.`film_id`,
8       `film`.`title`,
9       `film`.`description`,
10      `film`.`release_year`,
11      `film`.`language_id`,
12      `film`.`original_language_id`,
13      `film`.`rental_duration`,
14      `film`.`rental_rate`,
15      `film`.`length`,
16      `film`.`replacement_cost`,
17      `film`.`rating`,
18      `film`.`special_features`,
19      `film`.`last_update`
```

	film_id	title	description
▶	1	ACADEMY DINOSAUR	A Epic Drama of a Feminist And a Mad Scientist who must Battle a Teacher in The Canadian Rockies
▶	2	ACE GOLDFINGER	A Astounding Epistle of a Database Administrator And a Explorer who must Find a Car in Ancient China
▶	3	ADAPTATION HOLES	A Astounding Reflection of a Lumberjack And a Car who must Sink a Lumberjack in A Baloon Factory
▶	4	AFFAIR PREJUDICE	A Fanciful Documentary of a Frisbee And a Lumberjack who must Chase a Monkey in A Shark Tank
▶	5	AFRICAN EGG	A Fast-Paced Documentary of a Pastry Chef And a Dentist who must Pursue a Forensic Psychologist in
▶	6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler in Ancient China
▶	7	AIRPLANE SIERRA	A Touching Saga of a Hunter And a Butler who must Discover a Butler in A Jet Boat
▶	8	AIRPORT POLLOCK	A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India
▶	9	ALABAMA DEVIL	A Thoughtful Panorama of a Database Administrator And a Mad Scientist who must Outgun a Mad Scie

## film 2

## Output

## Action Output

Time	Action	Message	Duration / Fetch
4 16:12:02	SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE=TRADITIONAL'	0 row(s) affected	0.000 sec
5 16:12:02	CREATE TABLE actor ( actor_id SMALLINT UNSIGNED NOT NULL AUTO...	Error Code: 1050. Table 'actor' already exists	0.000 sec
6 16:13:59	SELECT film.film_id, film.title, film.description, film.release_ye...	1000 row(s) returned	0.031 sec / 0.000 sec

## SQL Additions

| SELECT

## Topic: SELECT

## Syntax:

```

SELECT
  [ALL | DISTINCT | DISTINCTROW]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT]
  [BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_C
ROWS]
  select_expr [, select_expr ...]
  [FROM table_references
    [PARTITION partition_list]
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}
    [ASC | DESC], ...
    [WITH ROLLUP]]
  [HAVING where_condition]
  [ORDER BY {col_name | expr | position}
    [ASC | DESC], ...
    [LIMIT {[offset],} row_count | row_c
offset]]
  [PROCEDURE procedure_name(argument_
    [INTO OUTFILE 'file_name'
      [CHARACTER SET charset_name]
      export_options
    | INTO DUMPFILE 'file_name'
      [INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]]
```

SELECT is used to retrieve rows selected from one or more tables, and can include UNION statements and subqueries, and Online help subqueries.

The most commonly used clauses of SELECT statement are:

- Each select\_expr indicates a column that you want to retrieve. There must be at least one select\_expr.
- table\_references indicates the table or tabl
- Starting in MySQL 5.6.2, SELECT supports selection using the PARTITION keyword with subpartitions (or both) following the name

Context Help Snippets

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- performance\_schema
- sakila**
  - Tables
    - actor
    - address
    - category
    - country
    - customer
    - film
    - film\_actor
    - film\_category
    - film\_text
    - inventory
    - language
    - new\_actors
    - payment
    - rental
    - cookies
    - staff
    - store
  - Views
  - Stored Procedures
  - Functions
- test

Management Schemas

Information

Table: **actor**

Columns:

<b>actor_id</b>	smallint(5) UN AI PK
<b>first_name</b>	varchar(45)
<b>last_name</b>	varchar(45)

Object Info Session

Query 1 store - Table actor

```

1 •   SELECT `actor`.`actor_id`,
2       `actor`.`first_name`,
3       `actor`.`last_name`
4   FROM `sakila`.`actor`;
5 •   SELECT * FROM sakila.actor;
  
```

Result Set Filter: E

actor_id	first_name	last_name
1	PENELOPE	GUINNESS
2	NICK	WAHLBERG
3	ED	CHASE
4	JENNIFER	DAVIS
5	JOHNNY	LOLOBRIGI...
6	BETTE	NICHOLSON
7	GRACE	MOSTEL
8	MATTHEW	JOHANSSON
9	JOE	SWANK
10	CHRISTIAN	GABLE
11	ZERO	CAGE
12	KARL	BERRY
13	UMA	WOOD
14	VIVIEN	BERGEN
15	CUBA	OLIVIER

CREATE INDEX Syntax

```

CREATE [UNIQUE|FULLTEXT|SPATIAL] INDEX index_name
[index_type]
ON tbl_name (index_col_name,...)
[index_option] ...

# index_col_name:
col_name [(length)] [ASC | DESC]

# index_type:
USING {BTREE | HASH | RTREE}

# index_option:
  
```

Edit Done

Context Help Snippets

SQL Additions

- SQL DDL**
- CREATE TABLE Syntax**
- CREATE VIEW Syntax**
- CREATE PROCEDURE / FUNCTION Syntax**
- CREATE INDEX Syntax**
- CREATE SCHEMA Syntax**
- ALTER TABLE Syntax**
- ALTER VIEW Syntax**
- ALTER PROCEDURE/FUNCTION Syntax**
- DROP TABLE Syntax**
- DROP VIEW Syntax**

# MySQL Workbench

## Administer

- MySQL Workbench provides a visual console to easily administer MySQL environments and gain better visibility into databases.
- Developers and DBAs can use the visual tools for configuring servers, administering users, performing backup and recovery, inspecting audit data, and viewing database health.

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

SQl SQL Databases Tables Scripts Reports EER Diagram

Navigator

**MANAGEMENT**

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

**INSTANCE**

- Startup / Shutdown
- Server Logs
- Options File

**MySQL ENTERPRISE**

- Audit Inspector
- Online Backup
- Backup Recovery

**SCHEMAS**

Filter objects

- information\_schema
- mysql
- performance\_schema
- sakila**
- test

Information

No object selected

Object Info Session

Ready

Query 1 MySQL Enterprise Backup - Ba...

## MySQL Enterprise Backup

Backup Profile Name: Production Backups Full Server / Full + Incremental

Comments:  
3 fulls and 4 incrementals for backups

Contents Options Schedule

Perform full backups every

Week on Sun Mon Tue Wed Thu Fri Sat at 02 : 00

Perform incremental backups every

Week on Sun Mon Tue Wed Thu Fri Sat at 02 : 00

Incremental backups will create a backup of all changes that have occurred since the lastest backup, full or incremental.

Note: backups are scheduled and executed from the target server, using the systems task scheduler as the user that owns the MySQL datadir.

Delete Save and Reschedule Cancel

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigator

**MANAGEMENT**

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

**INSTANCE**

- Startup / Shutdown
- Server Logs
- Options File

**MySQL ENTERPRISE**

- Audit Inspector
- Online Backup
- Backup Recovery

**Management Schemas**

Information

Connection:

- Name: *web\_database*
- Host: *127.0.0.1*
- Port: *3306*
- Server: *MySQL*
- Version: *5.6.12-enterprise-commercial-advanced*
- Login User: *root*
- Current User: *root@localhost*

Object Info Session

Query 1 Administration - Data Export

**web\_database Data Export**

Object Selection Export Progress

Select Database Objects to Export

Schema

- mysql
- performance\_schema
- sakila
- test

Refresh 24 tables selected

Advanced Options...

Exp... Schema Objects

- actor
- actor\_info
- address
- category
- country
- customer
- customer\_list
- film
- film\_actor

Select Tables Unselect All

Options

Export to Dump Project Folder C:\Users\mfrank\Documents\dumps\Dump20130724 ...

Each table will be exported into a separate file. This allows a selective restore, but may be slower.

Export to Self-Contained File C:\Users\mfrank\Documents\dumps\Dump20130724.sql ...

All selected database objects will be exported into a single, self-contained file.

Create Dump in a Single Transaction (self-contained file only)

Dump Events

Dump Stored Routines (Procedures and Functions)

Skip table data (no-data)

Press [Start Export] to start... Start Export

Management support for target host enabled successfully.

# MySQL Workbench

- **Database Migration**
- MySQL Workbench now provides a complete, easy to use solution for migrating Microsoft SQL Server, Microsoft Access, Sybase ASE, PostgreSQL, and other RDBMS tables, objects and data to MySQL.
- Developers and DBAs can quickly and easily convert existing applications to run on MySQL both on Windows and other platforms.
- Migration also supports migrating from earlier versions of MySQL to the latest releases.

# Database Technologies

Session -2

# Content

- Database Design, Entity-Relationship Diagram (ERD)
- Relation model
  - Database Constraints (Primary Key, Foreign Key, Candidate key)
- ER to Relation mapping
- Codd's 12 rules for RDBMS

# Database Design

# Database Design -Requirement

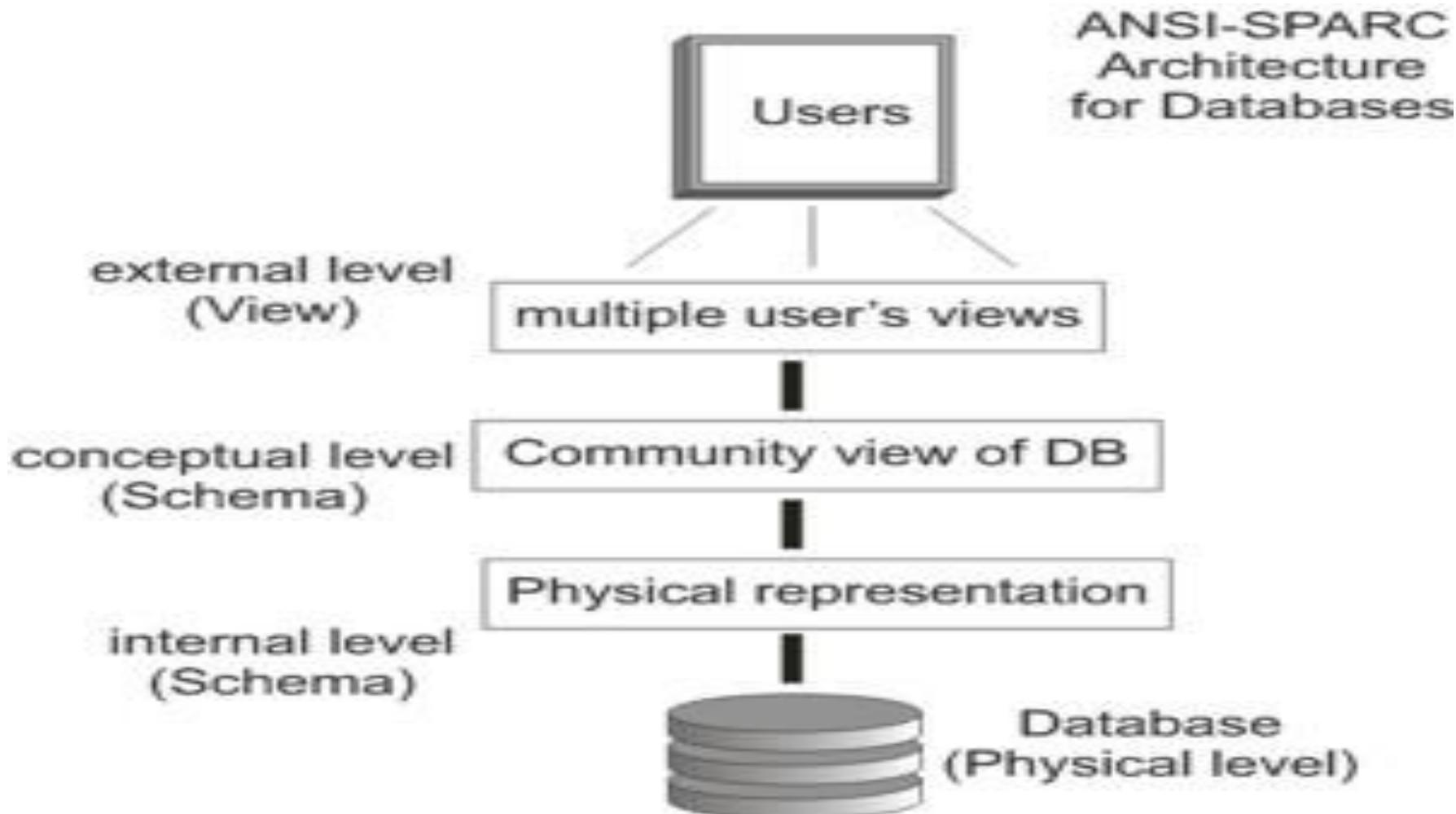
- Database should be easy to maintain
  - Storing only a limited amount (if any) of repetitive data.
  - If you have a lot of repetitive data and one instance of that data undergoes a change (such as a name change), that change has to be made for all occurrences of the data.
    - Create a master table

# The Conceptual Model

---

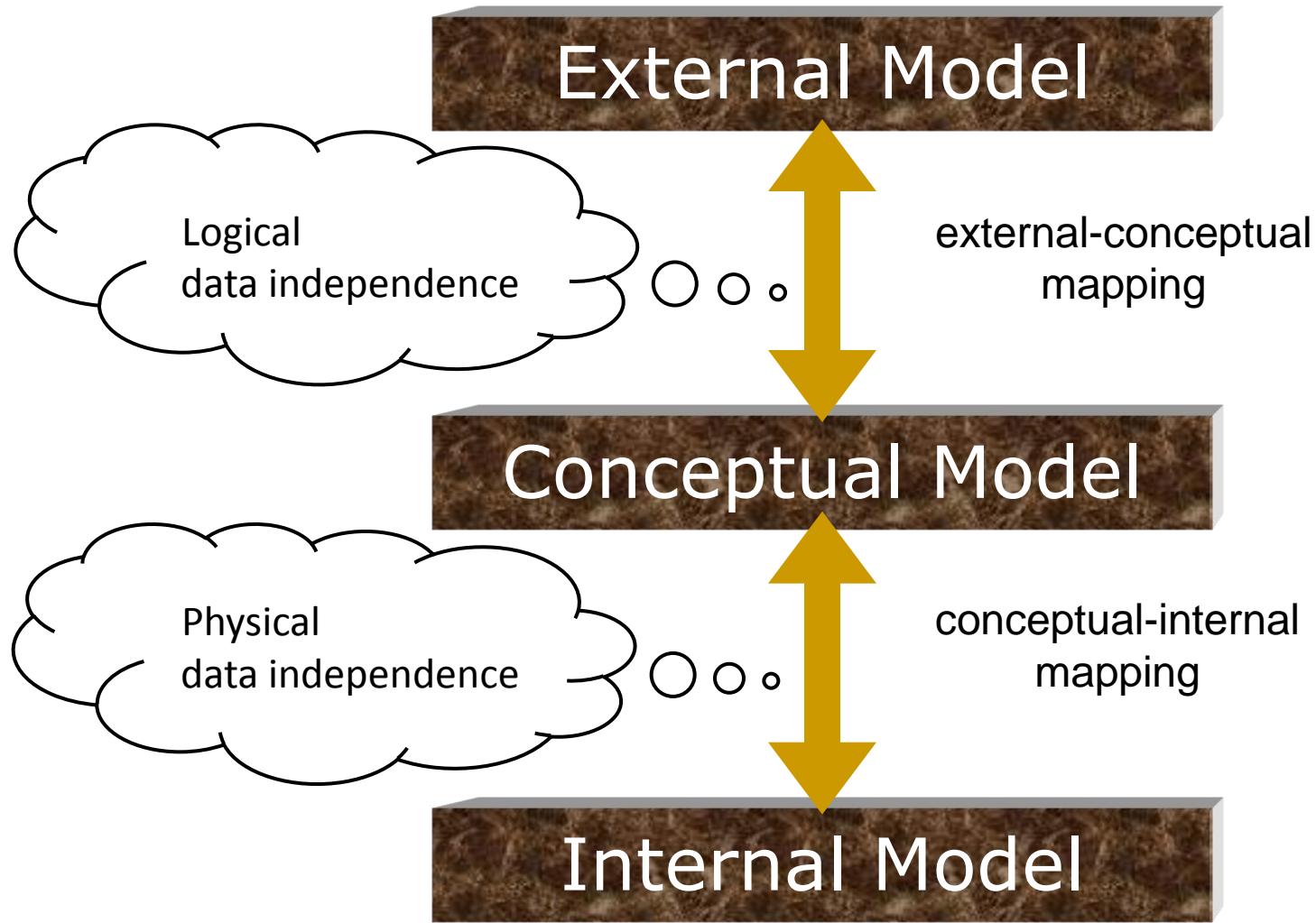
- ◆ ANSI/SPARC model advocates the 3-tier architecture - external model, conceptual model and the internal model.
- ◆ “conceptual model” captures the global/institutional view of the data semantics.
  - ◆ investigates and enumerates the various entities that participate in the business environment being modelled.

# Three Level Architecture



# ANSI/SPARC 3-Level Architecture

---



# Levels of Abstraction

- **Physical level:** describes how a record (e.g., customer) **is** stored.
- **Logical level:** describes data stored in database, and the relationships among the data.

```
type customer = record
    customer_id : string;
    customer_name : string;
    customer_street : string;
    customer_city : string;
end;
```

- **View level:** application programs **hide** details of data types. Views can also hide information (such as an employee's salary) for security purposes.

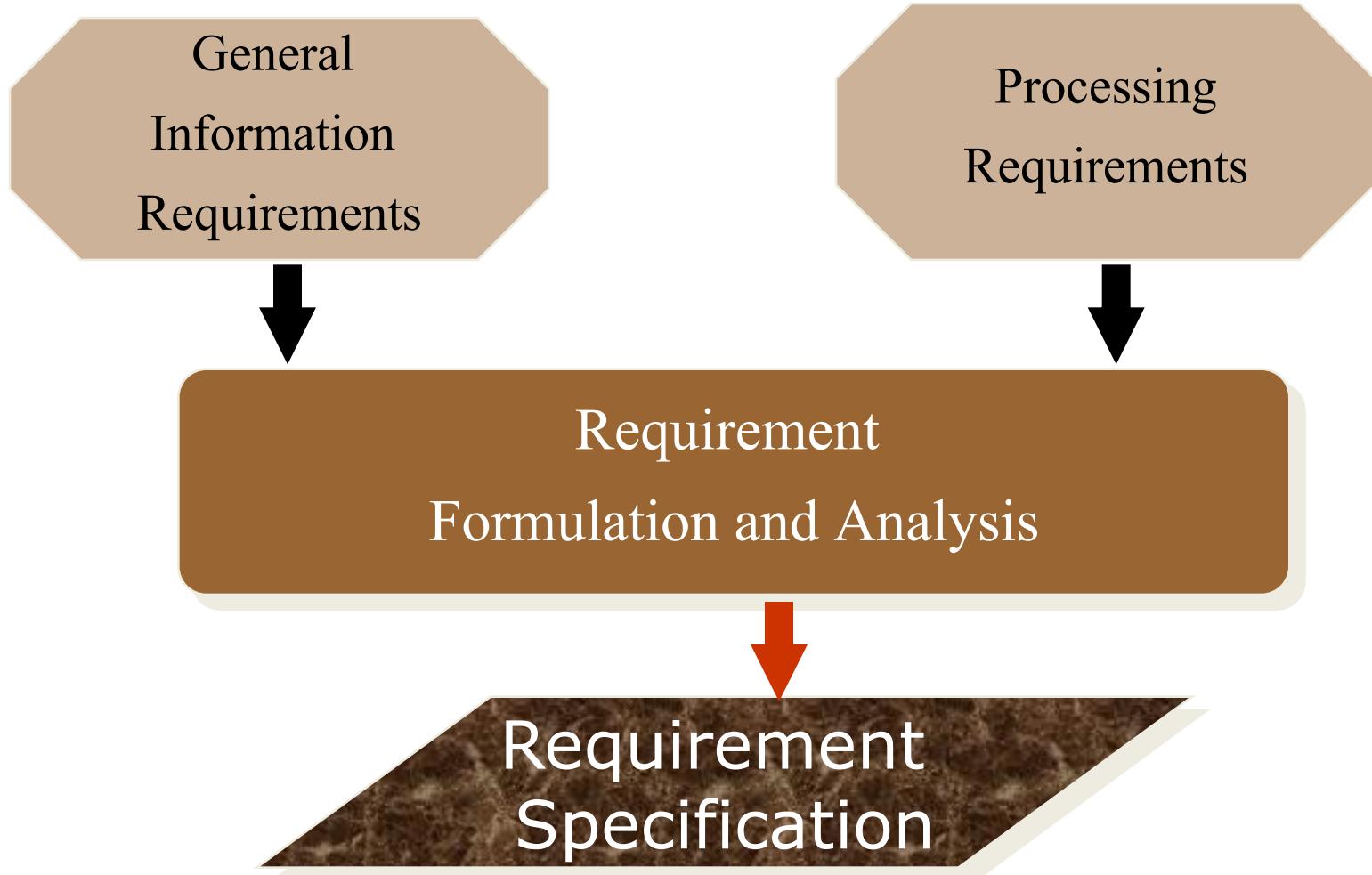
# Database Design

The process of designing the general structure of the database:

- Logical Design –
  - Deciding on the database schema.
  - Database design requires that we find a “good” collection of relation schemas.
  - Business decision –
    - What attributes should we record in the database?
  - Computer Science decision –
    - What relation schemas should we have and how should the attributes be distributed among the various relation schemas?
- Physical Design – Deciding on the physical layout of the database

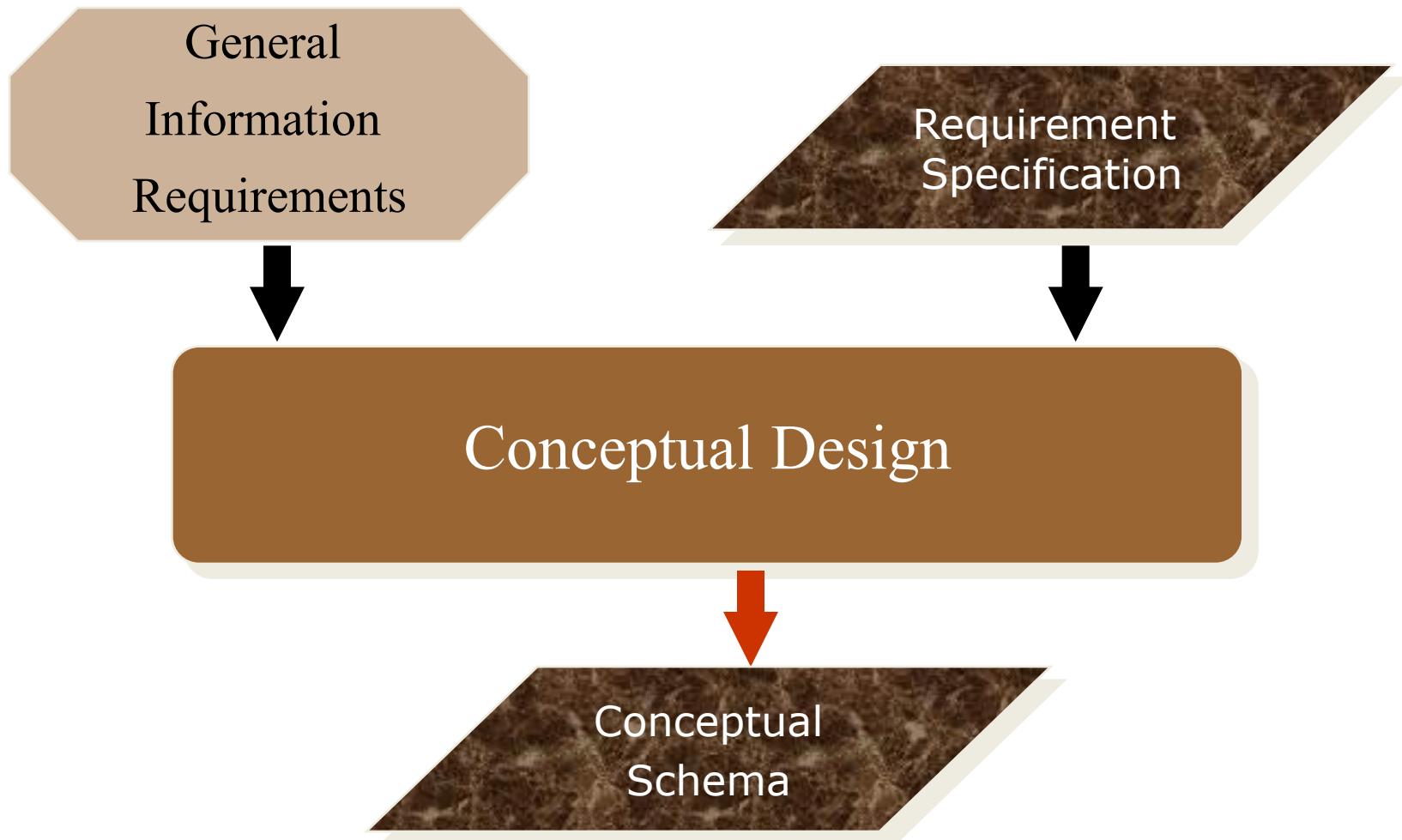
# Database Design Process

## Step 1



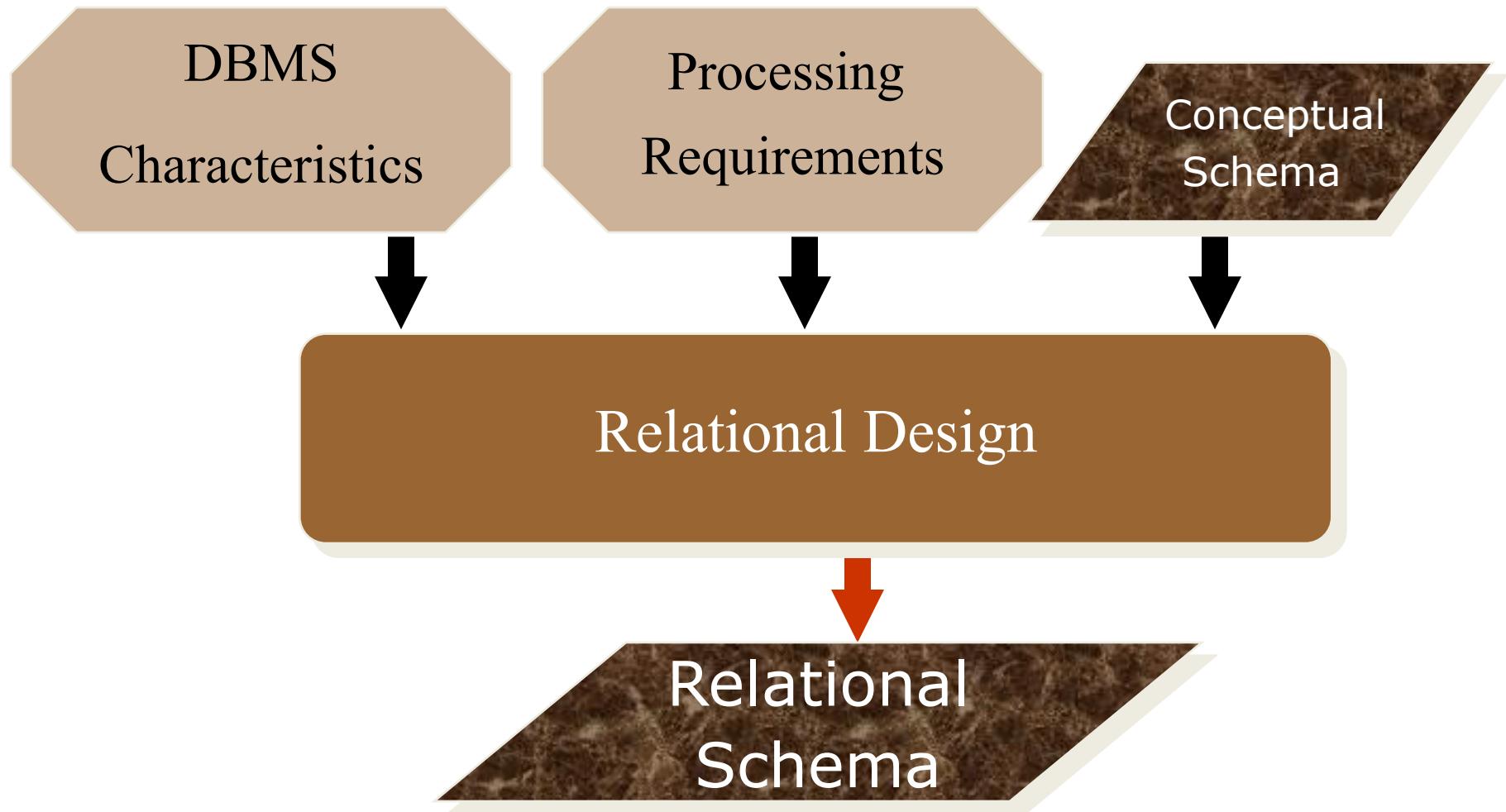
# Database Design Process

## Step 2



# Database Design Process

## Step 3



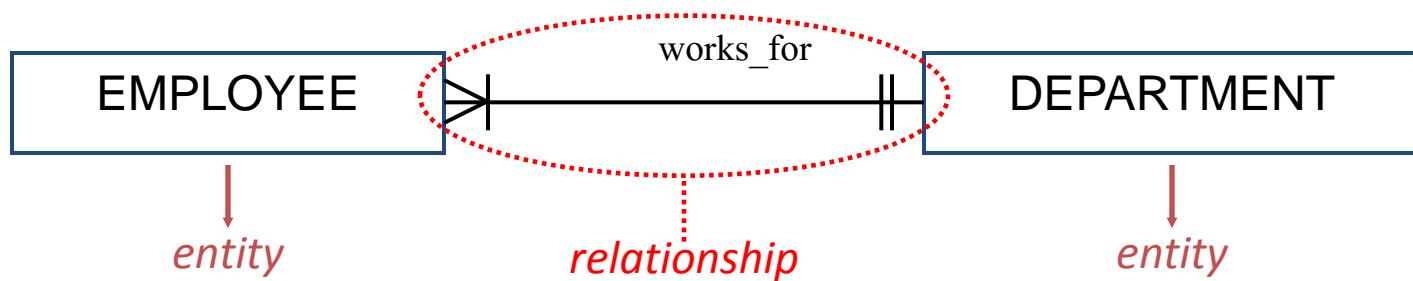
# ER Modeling

# E-R Modelling

---

- ◆ Entity-Relationship (E-R) Modelling is a conceptual modelling tool.
- ◆ perceives the business environment in terms of participating “entities” and the “relationship” between them.

*e.g. many employees work for a department.*



# The E-R Model

---

- ◆ “entity” represents some object of the real-world; “relationship” captures the association between entities (real-world objects).
- ◆ E-R model must capture all the business requirements, as well as the natural associations between business objects.
- ◆ must be complete and sound, but must be easily readable - to the designer as well as to the naive user.

# Building the ER Model

---

- ◆ the requirements specification is the first step to any design; it captures the ‘*what*’ of the business environment.
- ◆ also documents the “business rules” - i.e., the constraints that will apply to your database.  
*e.g. every department must have a manager;  
and only one manager.*
- ◆ the ER model must capture the participating entities as well as these business rules.

# Building the ER Model : Entity

---

- ◆ an “entity” (set) is a data object.
  - ◆ depicts a *set* of related/similar objects in the real world (not necessarily tangible).
  - ◆ usually identified by the nouns in the requirements specification.  
*e.g. ‘department’, ‘invoice’, ‘vehicle’, etc..*
-  not all nouns are entities, nor are all entities identified by nouns.

# Building the ER Model : Attributes

---

- ◆ “attributes” are properties of an entity.
- ◆ each instance (member) of an entity (set) will have the same attributes (properties), but different attribute *values*.

*e.g. “department” entity may have attributes  
“dept\_id” and “dept\_name”.*

*Every department will have these attributes; one department is differentiated from another by its id (value of “dept\_id”) and its name (value of “dept\_name”).*

# Building the ER Model : Relationship

---

- ◆ represents the real-world association between two or more entities; or even of an entity with itself.
- ◆ represents the association between entity *sets*; not between entity *instances*.
- ◆ are usually omni-directional; “roles” may be used where required, to add meaning.
- ◆ a “role” is a name (usually a noun) of one end of a relationship.

# Building the ER Model : Relationship

---

- ◆ a role indicates the function of an individual entity in the relationship.
- ◆ relationships are usually identified by the verbs in the requirements specification.

e.g. “*employee works for a department*”.

entities: “*employee*”, “*department*”

relationship: “*works\_for*”

- ❑ not all verbs are relationships; nor are all relationships identified by verbs.

# Building the ER Model : a recap

---

- ◆ the requirements specification is the stepping stone to an ERD.
- ◆ typically, the nouns identify the entities, and the verbs identify the relationships.
- ◆ entities are defined in terms of its attributes; which serve to capture its real-world properties.
- ◆ relationships denote associations, and therefore capture business rules (constraints).

# Building the ER Model : an example

---

ABC Traders is a trading concern that is a distributor for several categories of products, and sells numerous products of each category.

Apart from its fixed retail customers (retail outlets), ABC Traders also have counter sales at their sales depot.

Fixed retail customers are extended an agreed period of credit and to an extent specified. Counter sales are strictly cash sales.

ABC employs several salesmen who are responsible for collecting orders from the customers and to follow-up on that order - delivery as well as payment.

# Building the ER Model : example

A given order can be for several products, but will be for only one category of products.

In case of non-availability of an ordered product, no back-order is generated.

Alternatively, a customer can cancel an order he has placed, notwithstanding availability of goods.

No advance is collected along with the order; full payment is expected after invoicing and within the specified credit period.

Part payments are not allowed, but a customer can pay several invoices in one payment. Payment may either be in cash or through cheque or draft.

# Building the ER Model : example

Potential Entities:

~~ABC Traders~~  
~~trading concern~~  
~~distributor~~  
categories  
products  
retail customers  
~~sales depot~~  
? credit period  
salesmen

order  
back-order  
invoice  
availability  
~~delivery~~  
? payment  
advance  
cash  
cheque  
draft

# Building the ER Model : example

Attributes: (some sample cases)

- ◆ CUSTOMER

- cust\_id
- cust\_name
- cust\_class
- cust\_contactperson
- cust\_address
- cust\_city
- cust\_tel1
- cust\_tel2
- cust\_fax
- cust\_email

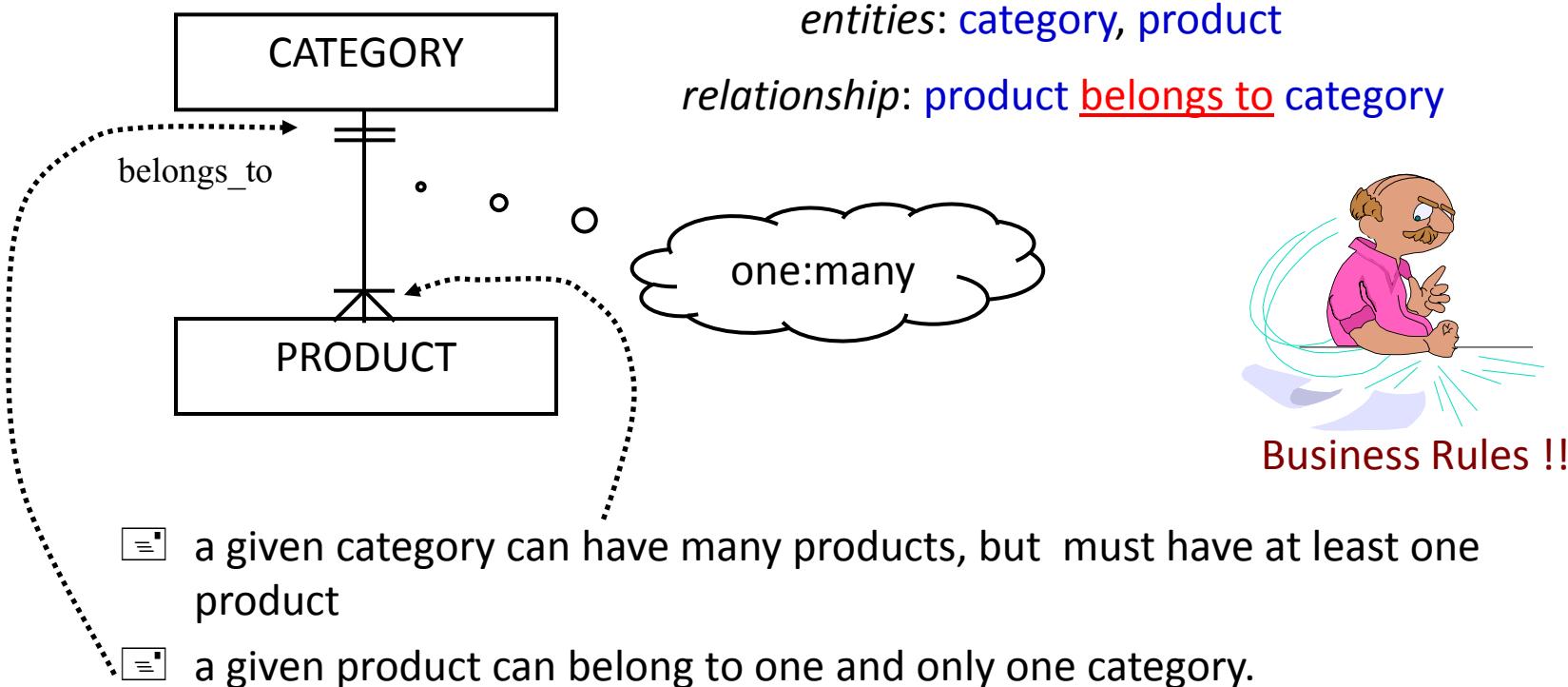
- ◆ SALESMAN

- slman\_id
- slman\_name
- slman\_birthdt
- slman\_joindt
- slman\_address
- slman\_city
- slman\_tel
- slman\_fuelallowance

# Building the ER Model : example

Relationships: (*a sample case*)

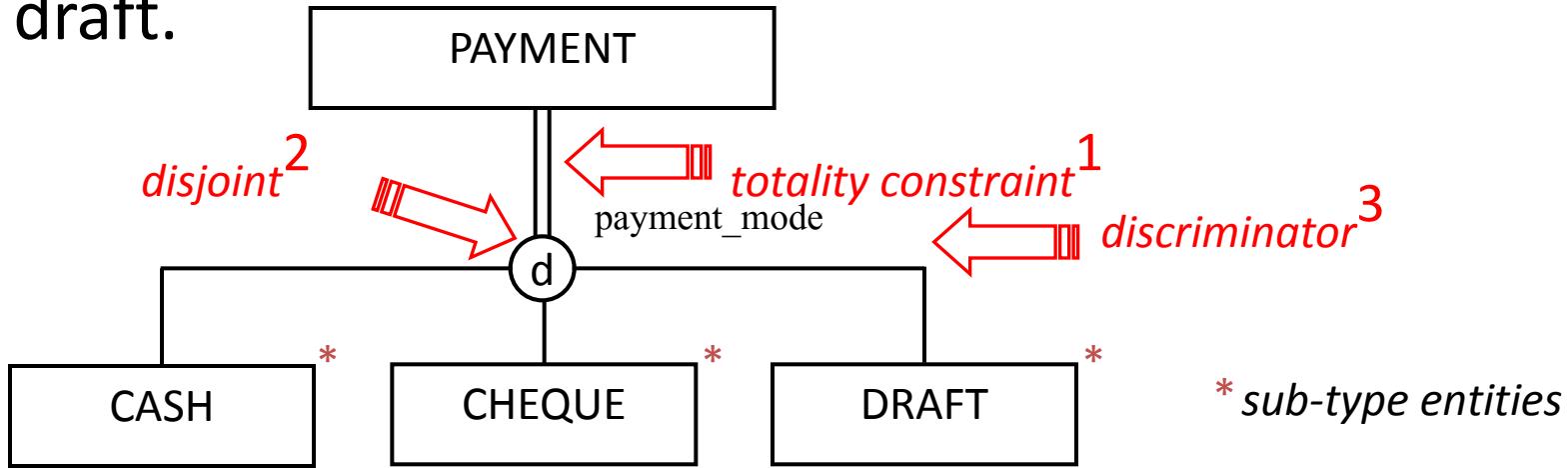
ABC sells many categories of products;  
each category has several products



# Building the ER Model : example

## Advanced Constructs (*Specialisation*)

Customers pay their invoices in cash, cheque or draft.



<sup>1</sup> every instance of *payment* has to be of *at least* one sub-type

<sup>2</sup> an instance of *payment* can be of *exactly* one sub-type

<sup>3</sup> the criteria on which an instance of *payment* is classified into one of the sub-types.

# Building the ER Model : example

Invoices are delivered to  
customers “delivery” : attribute or entity ??



- customer signs duplicate invoice; only store whether invoice is delivered or not
- a given invoice is delivered at most, once.
- delivered items are same as invoice items

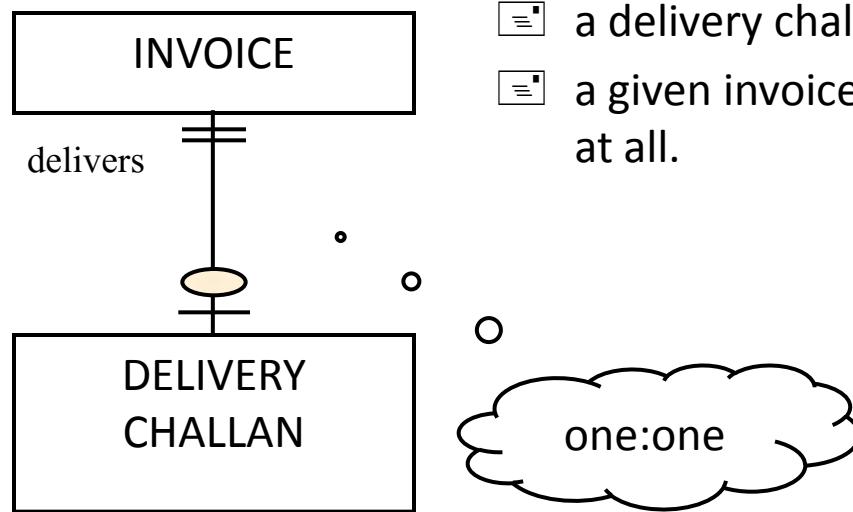
- acknowledge delivery
- record delivery challan number
- record delivered items
- record mode of delivery



# Building the ER Model : example

ER modelling : attribute or entity ??

Invoices are delivered to customers.



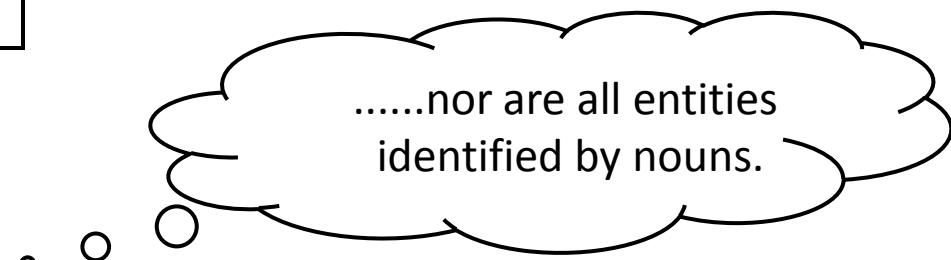
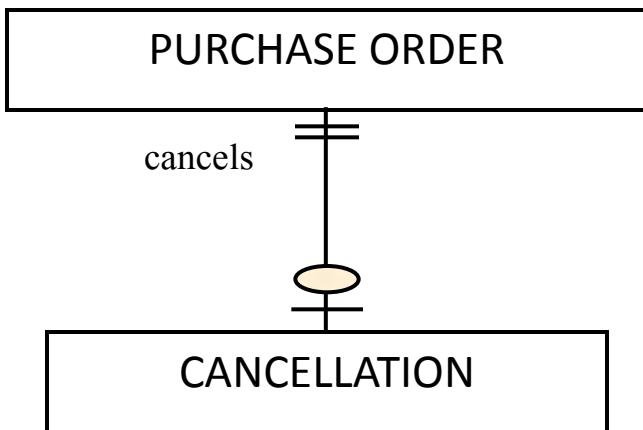
- a delivery challan delivers one and only one invoice
- a given invoice is delivered entirely or not delivered at all.



*Generally, in the case one:one relationships, one entity can be “collapsed” into the other entity and modelled as attributes of the other entity.*

# Building the ER Model : example

Alternatively, a customer can cancel an order.....

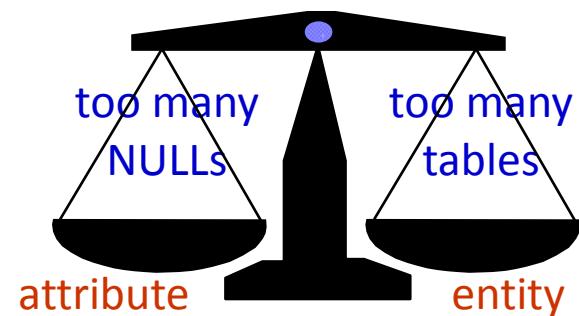


*“cancellation”, “delivery” : Attribute or entity ?*

## ☒ Business Rules

- ⇒ what is the usual practice ?
- ⇒ what is the data ?
- ⇒ how much data ?
- ⇒ retrieve how often ?

## ☒ Performance tuning issues



# Entity : Properties

---

An entity must exhibit the following properties:

- ◆ lies within the scope of the business world being modelled
- ◆ represents a set of similar objects *about which the enterprise needs to store information.*
- ◆ provides the ability to distinguish between various instances of the entity
- ◆ satisfies the rules of “normalization”.

# Entity : Types

---

Entities may be categorised on the basis of (common) characteristics into:

- ◆ fundamental/strong entity
  - an entity that is capable of its “own existence” - i.e. an entity whose instances exist notwithstanding the existence of other entities.
- ◆ weak entities
- ◆ associative entities
- ◆ sub-type entities

# Attributes

---

- ◆ characteristics/properties of an entity, that provide descriptive details of it.
- ◆ an entity can thus be thought of, as an ordered set of attributes.
- ◆ every *instance* of an entity is then, merely an ordered set of attribute *values*.
- ◆ each attribute is associated with a set of possible values; this set is called a “*domain*”.

# Attributes : Identifier

---

- ◆ an attribute that takes unique values, such that it can uniquely *identify* an entity instance.  
*e.g. “dept\_id” in “department” entity*
- ◆ an identifier is also known as a “key”.
- ◆ need not comprise only one attribute, can have two or more attributes (*composite key*)
- ◆ no attribute can identify more than one entity except in a “*specialisation hierarchy*”.

# Attributes : values

---

- ◆ the rules of “normalisation” dictate that every attribute shall take atomic values; multi-valued attributes are generally not allowed.
- ◆ multi-valued attributes suggest an entity by itself.
- ◆ further constraints can be imposed on attribute values (apart from the domain)  
*e.g. “not null” prevents an attribute from taking nulls.*

# Relationship

---

- ◆ models the real-world association between two or more entities (*binary, n-ary relationship*).
- ◆ may also model the association of an entity with itself (*recursive relationship*).
- ◆ must have a name that is unique across the entire model.
- ◆ must wherever possible, be supported by a well-documented description.

# Relationship : Properties

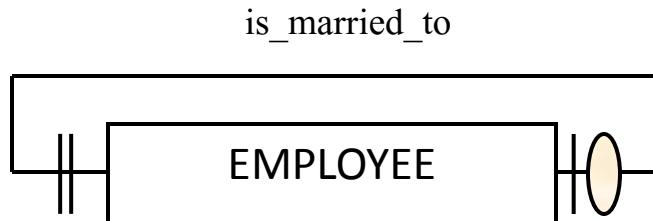
---

- ◆ described in terms of “*degree*”, “*connectivity*”, and “*existence*”.
- ◆ translates into “*migration of the key*” as follows:
  - ⇒ in the case of one:one relationships, from either entity to the other.
  - ⇒ in the case of one:many relationships, from the entity at the “one” end to that at the “many” end.
  - ⇒ in the case of many:many relationships, from both entities “into the relationship”

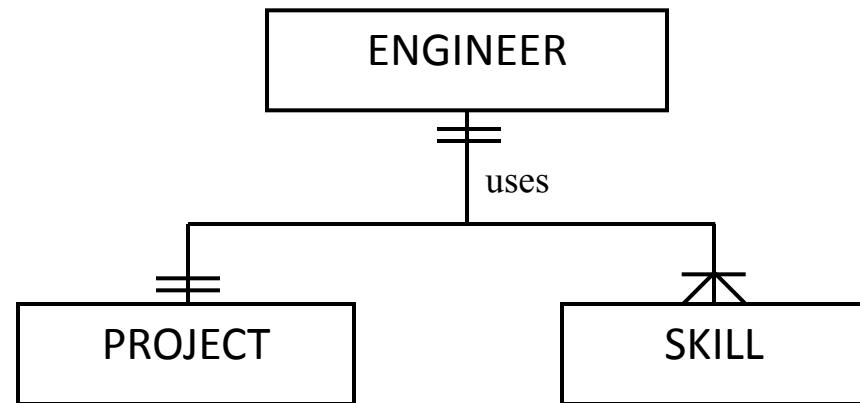
# Relationship : Degree

---

- ◆ “degree” describes the number of entities involved in the relationship.
- ◆ typically 2 (binary); other common degrees are 1 (recursive) and 3 (ternary).



*Recursive*



*Ternary*

# Relationship : Connectivity

---

- ◆ “connectivity” indicates the entity *occurrences* (instances) participating in a relationship.
- ◆ takes values “one” or “many”.  
*e.g. a one:many relationship indicates that for every occurrence of one entity, there are many related instances of the other entity.*
- ◆ an actual count of this connectivity (instead of “one” & “many”) if specified, is called the *cardinality* of the relationship.

# Relationship : Existence

---

- ◆ “existence” defines whether the relationship is optional or mandatory.
- ◆ determined by business rules.
- ◆ existence implicitly defines the minimum connectivity of a relationship ('0' if optional, '1' if mandatory).

*e.g. a project has to be managed by one employee,  
but not every employee manages a project.*

# Relationship : other issues

---

- ◆ a relationship name is usually read from left (entity) to right (entity) in an ER diagram; or from top to bottom\*.

*e.g. entities are “customer” and “order”.  
relationship is “places”.*

*In the ERD, place “customer” to the left of “order”.  
The construct will then be read as  
“customer *places* order”.*

\* applicable only to crow's-foot and IDEF1X notation

# Entity types : Weak

---

- ◆ an entity that is *not* capable of “its own existence”.
- ◆ characterised by the need to have at least one external identifier (of another entity) as part of its own identifier.

*e.g. consider “payment” and “pmt\_items”*

*“pmt\_items” cannot exist without a corresponding*

*“payment” instance. “pmt\_id” of “payment” will be part of the identifier of “pmt\_items”*

# Entity types : Associative

---

- ◆ a relationship translates into migration of a key - many:many relationship implies the keys migrating many times, *both ways*.
- ◆ such migration leads to redundancy and many:many relationships must therefore be resolved.
- ◆ “Associative entity” is an entity that is used to resolve a many:many relationship.

# Entity types : Associative

---

- ◆ also called “*intersecting entity*”.
- ◆ identifier of an associative entity is a composite of the identifiers of the entities involved in the many:many relationship.
- ◆ associative entities by their very definition, are also weak entities.
- ◆ must be appropriately named

# Entity types : Sub-type

---

- ◆ part of a specialisation hierarchy.
- ◆ categorise a subset of the occurrences of the parent entity.
- ◆ has (inherits) all the attributes of the parent entity, but also has additional attributes that are specific to that subset of occurrences.
- ◆ every entity must be associated with at least one super-entity.

# Entity types : Sub-type

---

- ◆ specialisation can either be *total* (every instance of super-type must belong to at least one sub-type) or can be *partial*.
- ◆ specialisation can either be *disjoint* (every super-type instance can belong to at most, one sub-type) or can be *overlapping*.

# The Crow's-Foot Notation

---

## Entity



## Connectivity & Existence

one, mandatory

*(one and only one)*



one, optional

*(zero or one)*



many, mandatory

*(many, but at least one)*



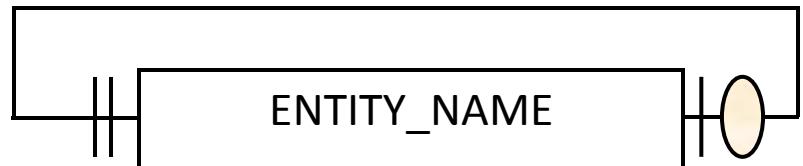
many, optional

*(zero, one or more)*

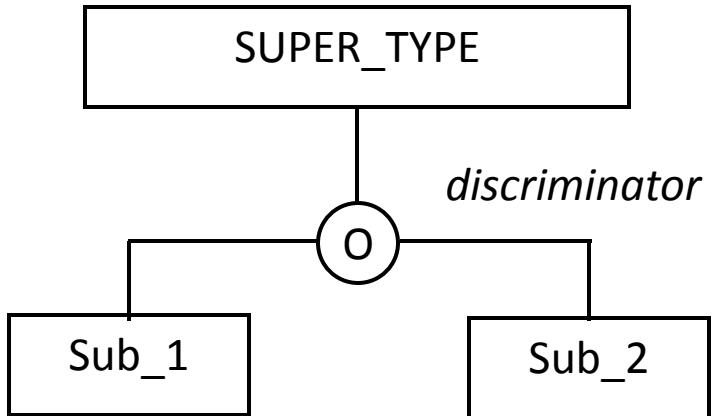


## Recursive Relationship

has\_relationship\_with



## Specialisation Hierarchy



# The Chen Notation

---

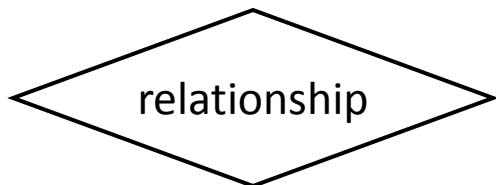
## Entity



## Cardinality

⇒ 'M' and/or 'N'  
⇒ '1'

## Relationship

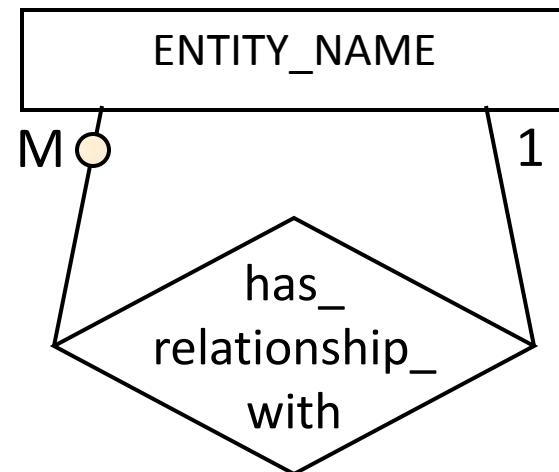


## Existence

optionality

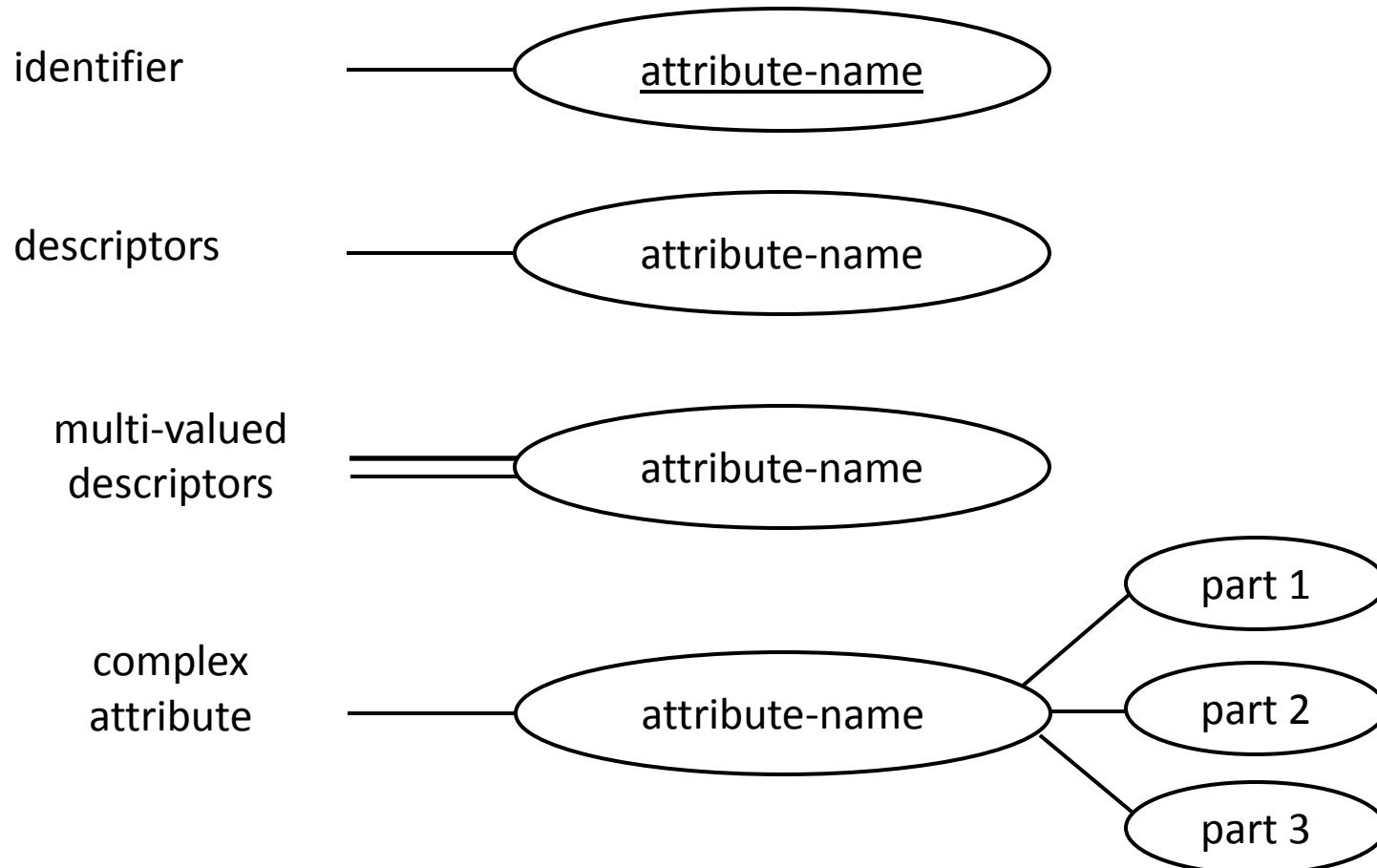


## Recursive Relationship



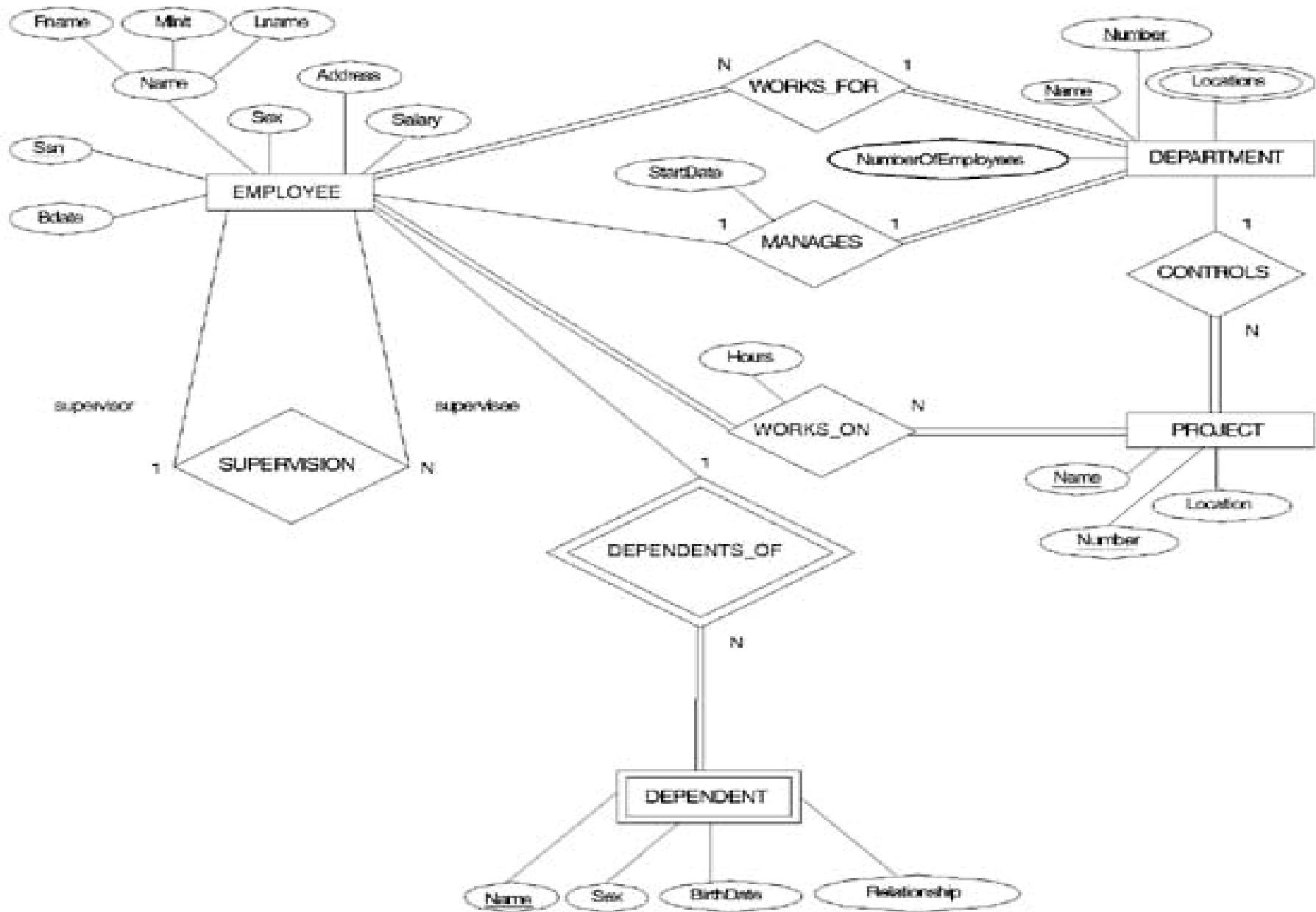
# The Chen Notation : Attributes

## Attributes



# SUMMARY OF ER-DIAGRAM NOTATION FOR ER SCHEMAS

<u>Symbol</u>	<u>Meaning</u>
	ENTITY TYPE
	WEAK ENTITY TYPE
	RELATIONSHIP TYPE
	IDENTIFYING RELATIONSHIP TYPE
	ATTRIBUTE
	KEY ATTRIBUTE
	MULTIVALUED ATTRIBUTE
	COMPOSITE ATTRIBUTE
	DERIVED ATTRIBUTE
	TOTAL PARTICIPATION OF E <sub>2</sub> IN R
	CARDINALITY RATIO 1:N FOR E <sub>1</sub> :E <sub>2</sub> IN R
	STRUCTURAL CONSTRAINT (min, max) ON PARTICIPATION OF E IN R



# **ER to Relational Mapping**

---

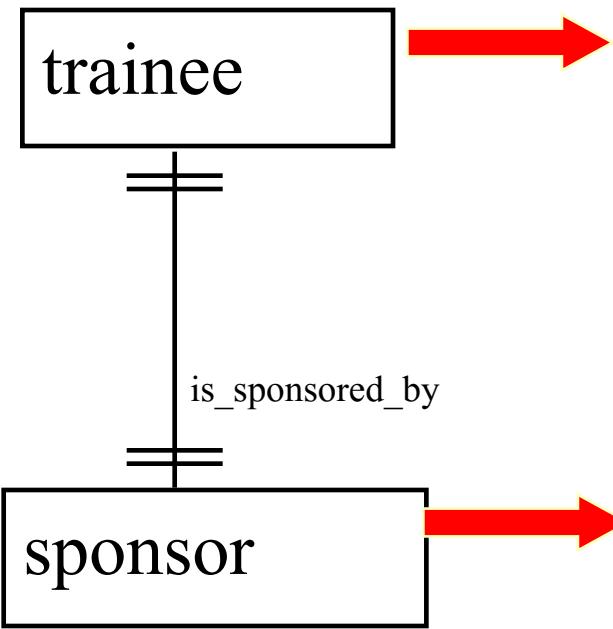
# Translating the E-R Model

---

- ◆ E-R model captures the database at the conceptual level - translating the model into the relational model (or any other model) yields the actual database.
- ◆ a Data-Definition Language (DDL) is used to set up the actual database schema.

# One-to-One Mandatory

Every trainee has one and only one sponsor and every sponsor sponsors one and only one trainee.



Empld	.....	....	Sp_Id

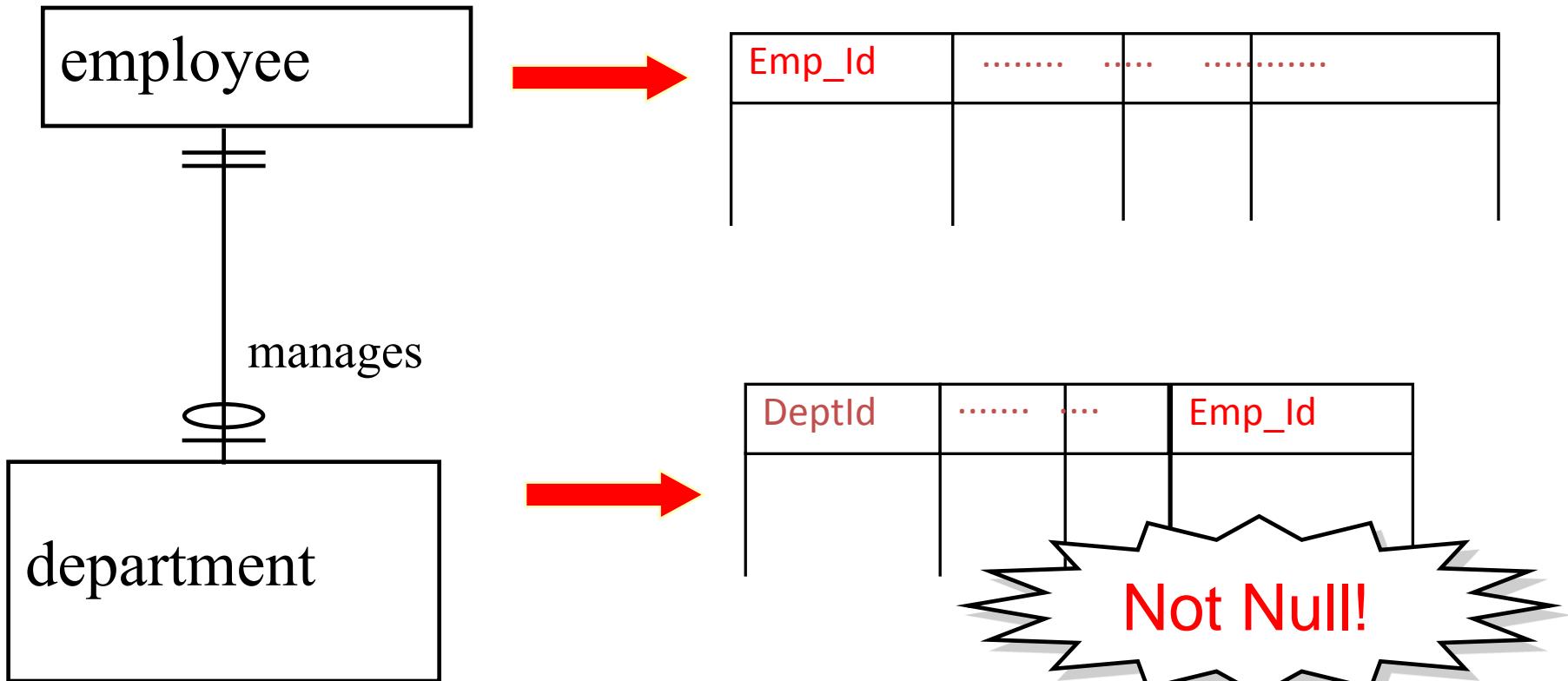
**Not Null!**

Sp_Id	.....	....	.....

- In a one:one relationship, one entity can usually be modelled as an attribute of the other entity.
  - For example, the entire 'sponsor' entity could be attributes of the 'trainee' entity (or vice-versa).
- When one of these two entities ALONE has an association with some other entity, then the two entities may be modelled separately.

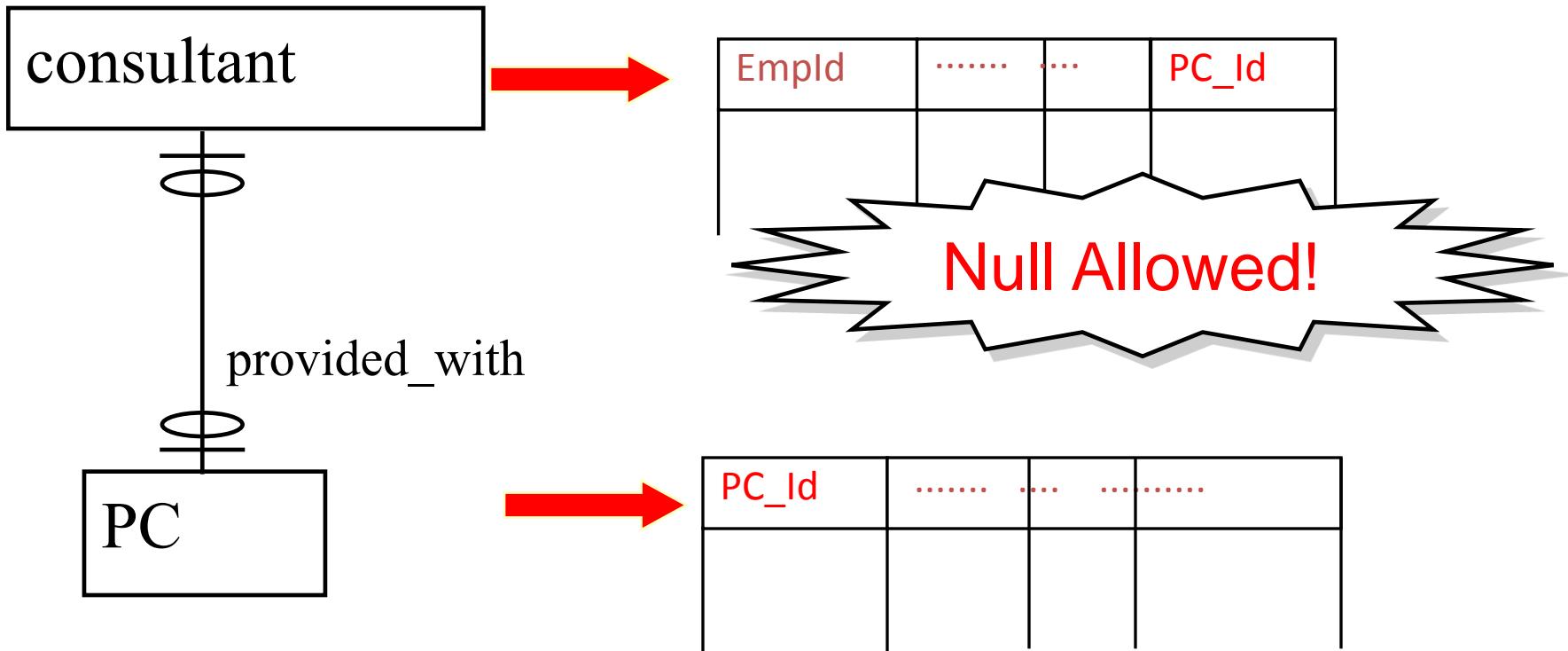
## One-to-One Optional

Every dept has one and only one manager; an employee can be the manager of at most one dept.



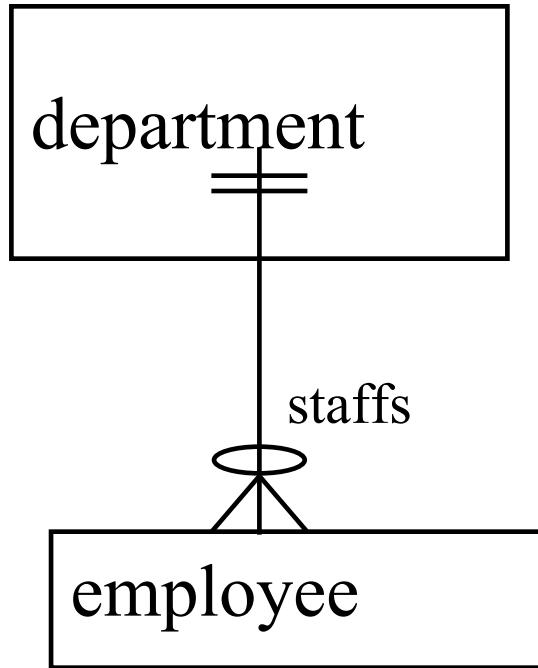
# One-to-One Optional

Some of the consultants are provided PCs at their desks.



## One-to-Many Optional

Every employee is allotted a department. A department however, staffs none, one or many employees.



Dept	.....	.....	.....

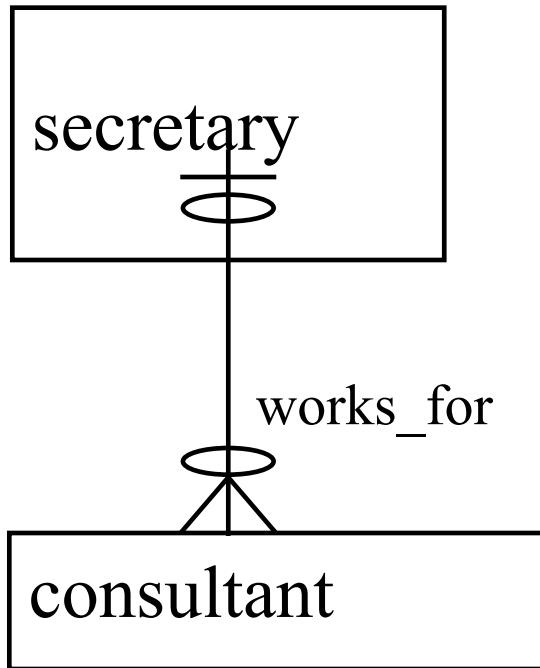


Empld	.....	.....	Dept

Not Null!

## One-to-Many Optional

A consultant can have at most one secretary. A secretary may work for several consultants.



Sec_Id	.....	....	.....

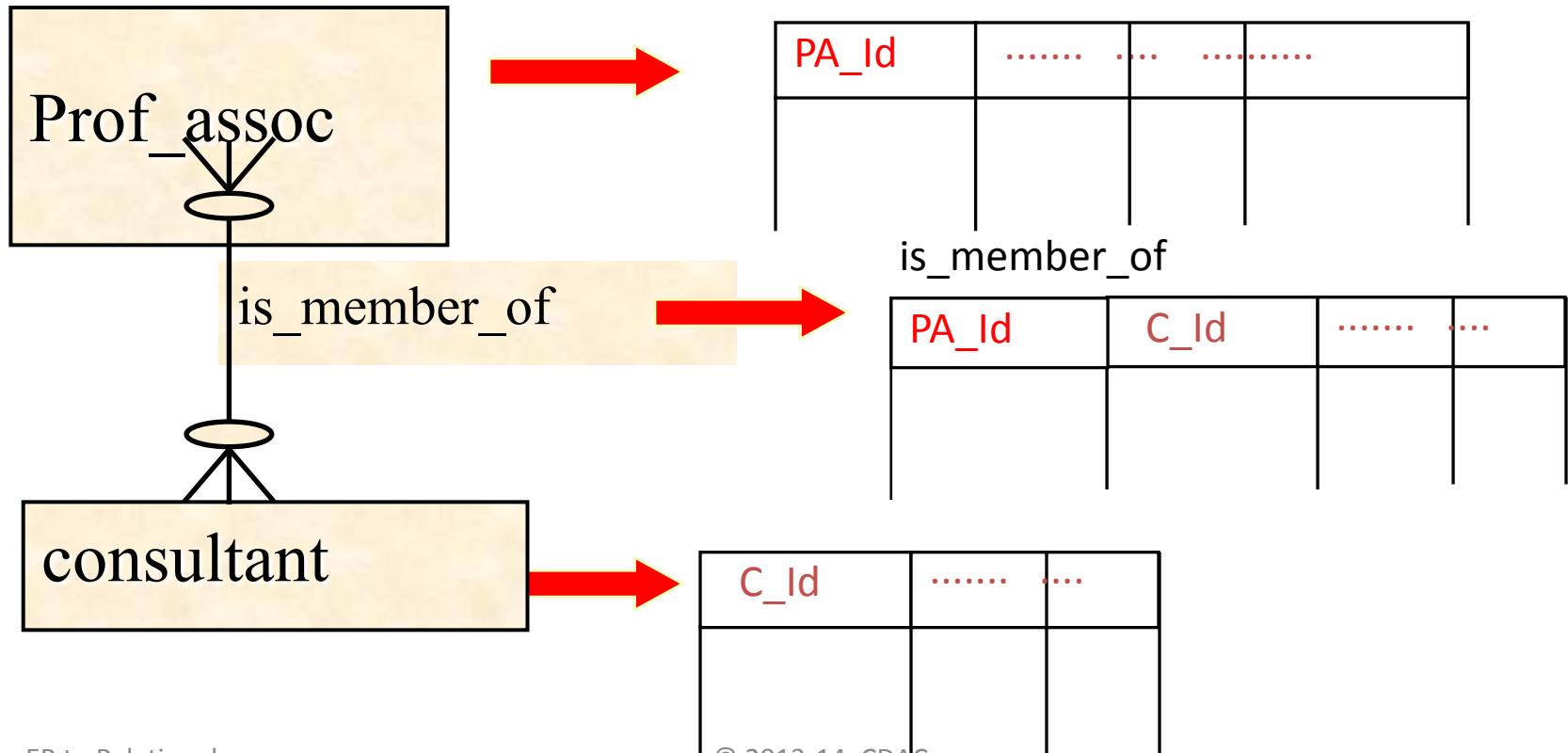


C_Id	.....	....	Sec_Id

Null Allowed!

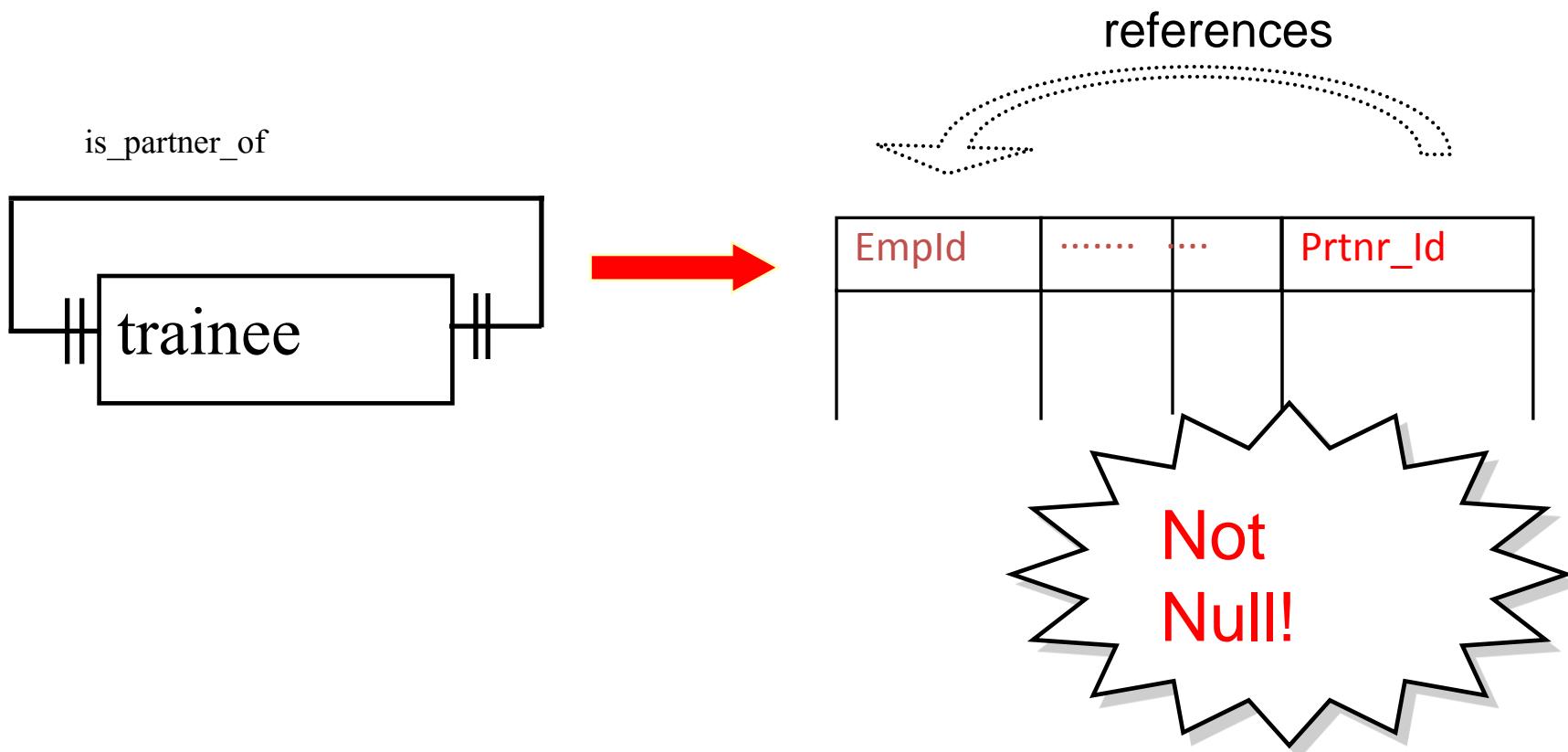
# Many-to-Many Optional

A professional association may have none, one or more consultants as members. Similarly, a consultant may be a member of none, one or many professional associations.



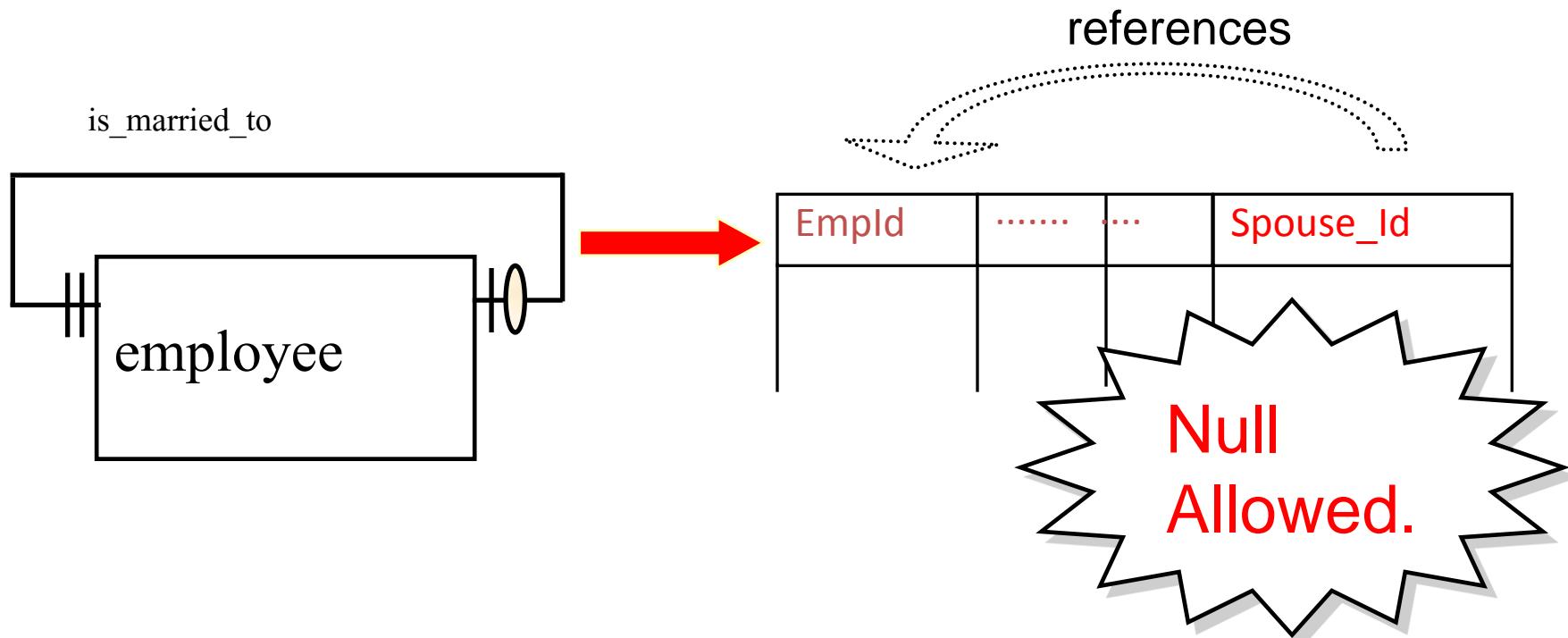
# Recursive One-to-One Mandatory

Every trainee has another trainee as a partner.



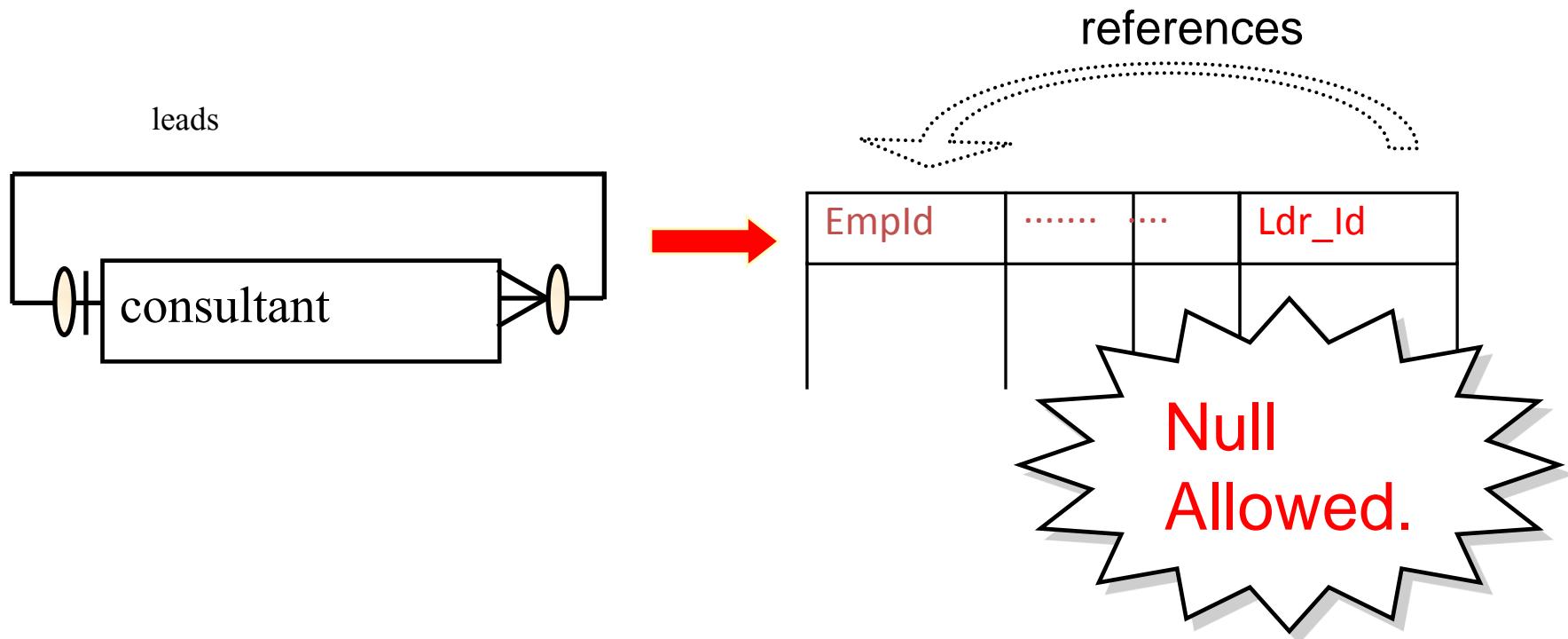
# Recursive One-to-One Optional

An employee may have another employee as a spouse.



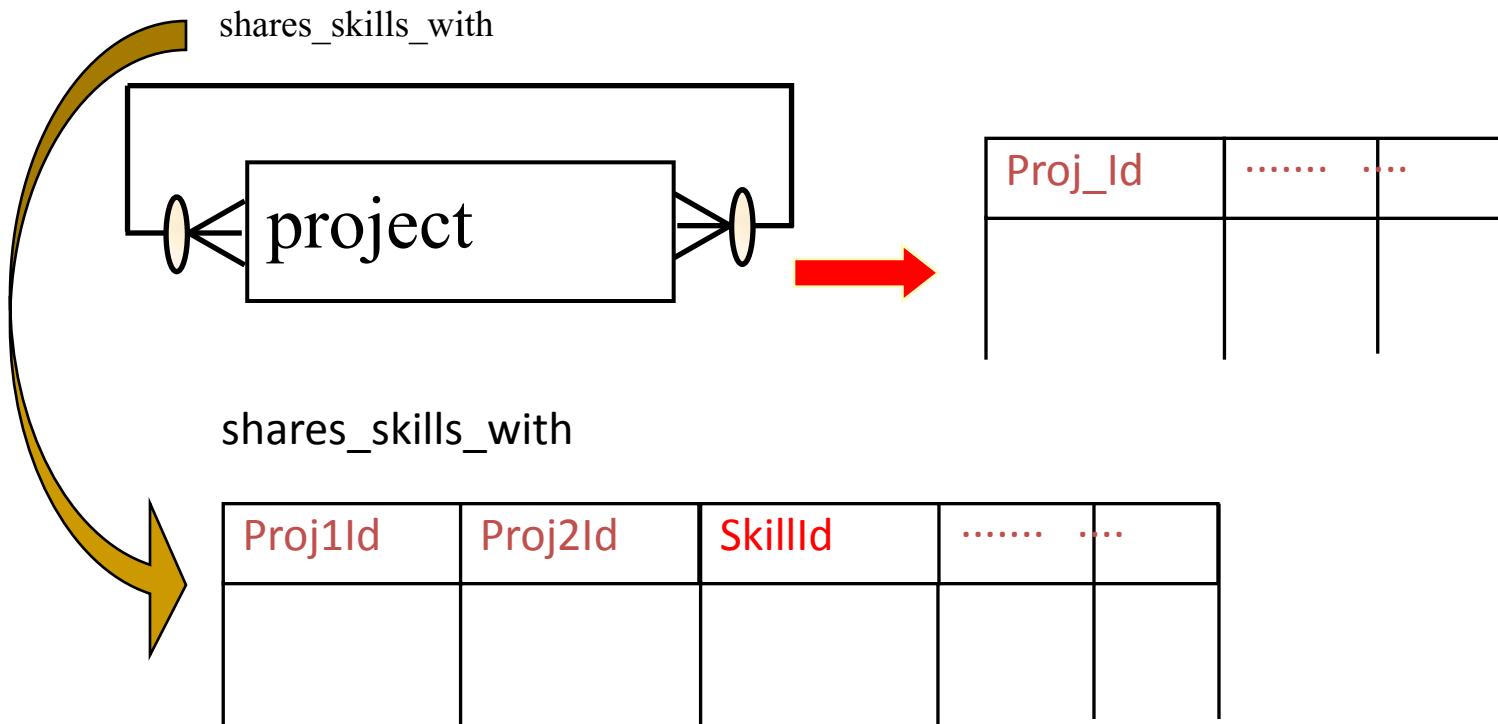
Recursive  
One-to-Many  
Optional

Consultants are divided into groups with each group having a leader.



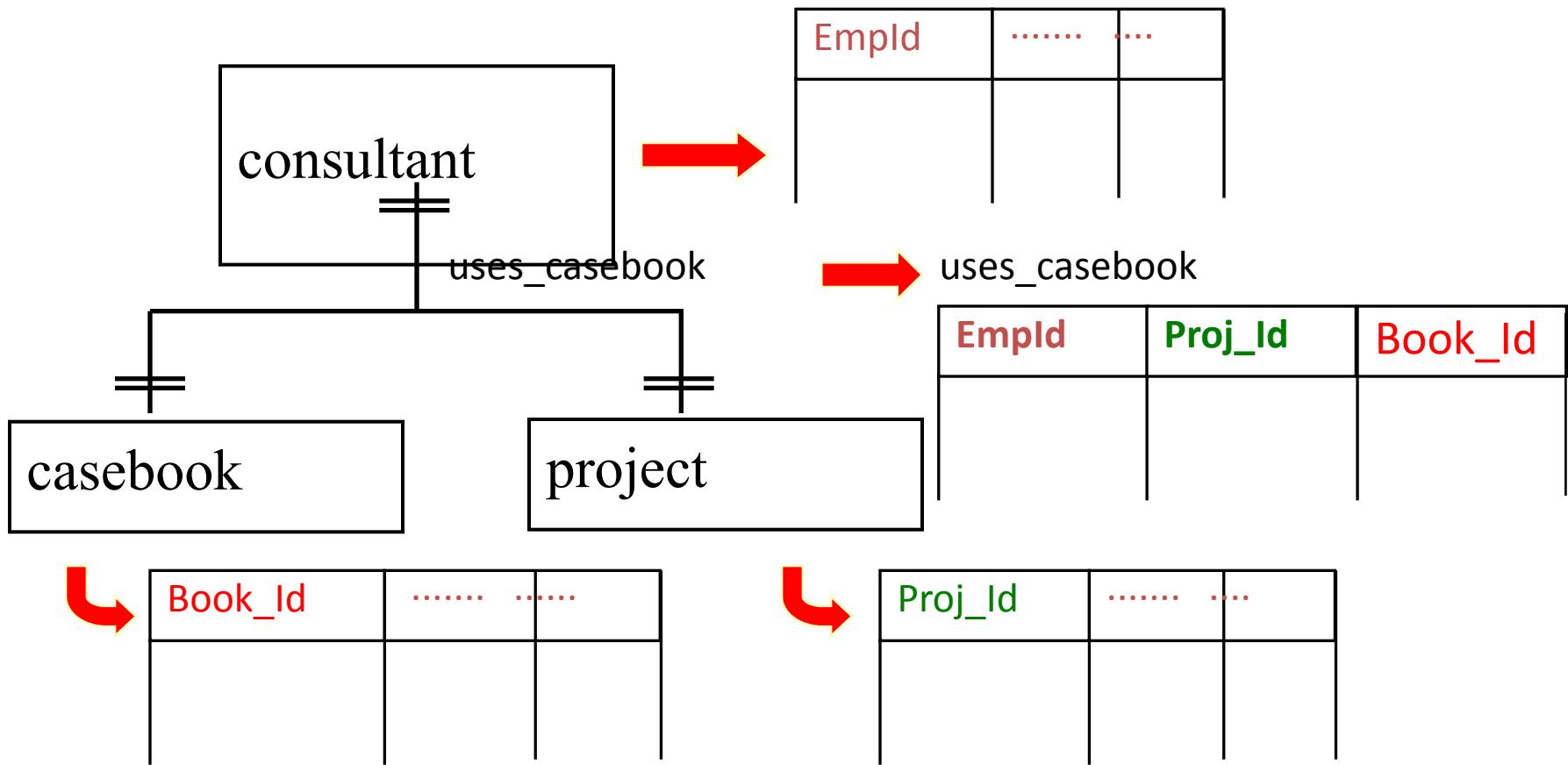
# Recursive Many-to-Many Optional

Several projects employ similar skill sets.  
i.e., projects share skills.



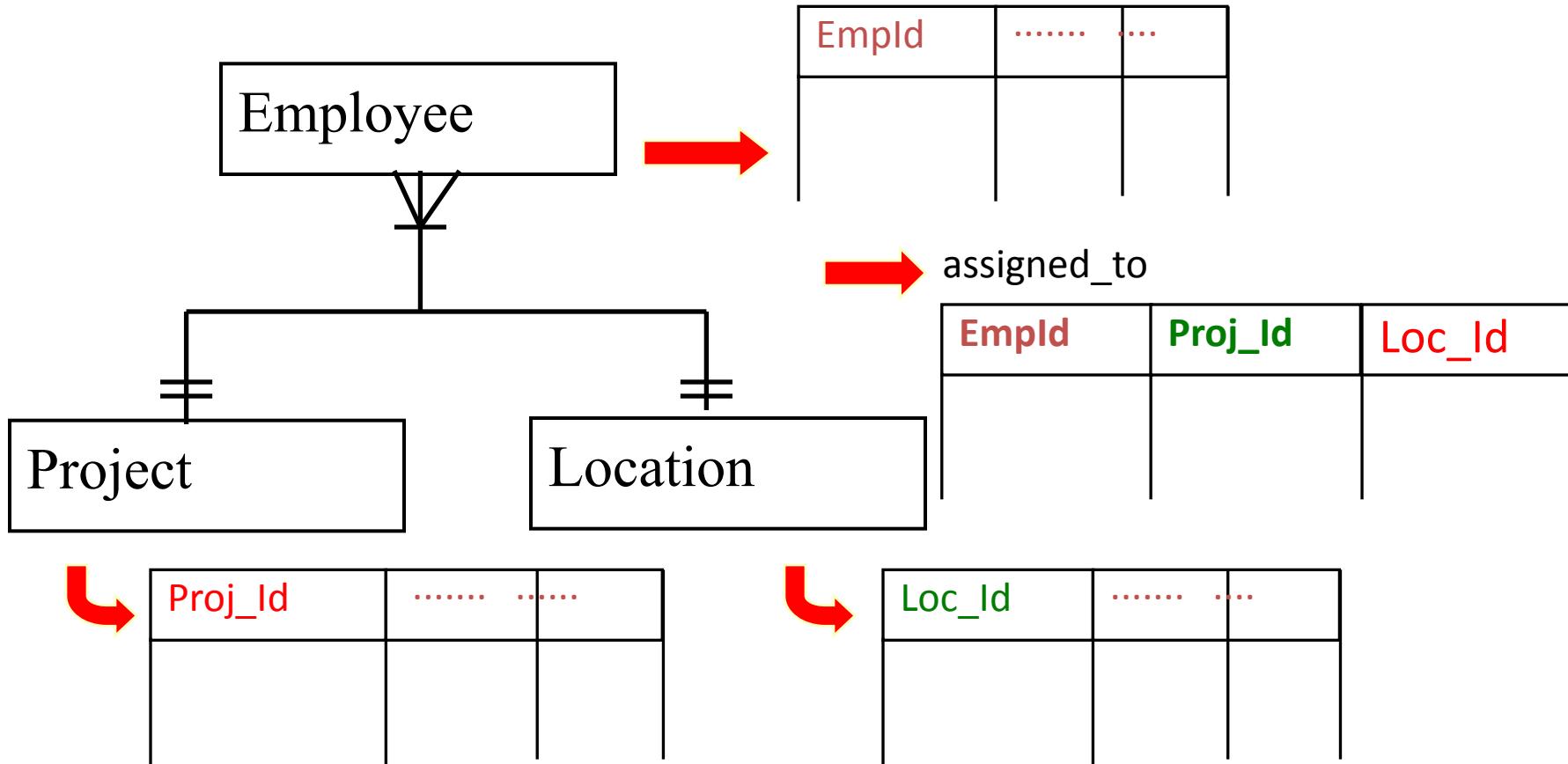
# Ternary One-One-One Mandatory

Consultants use casebooks for projects. Each consultant uses a different casebook for different projects.



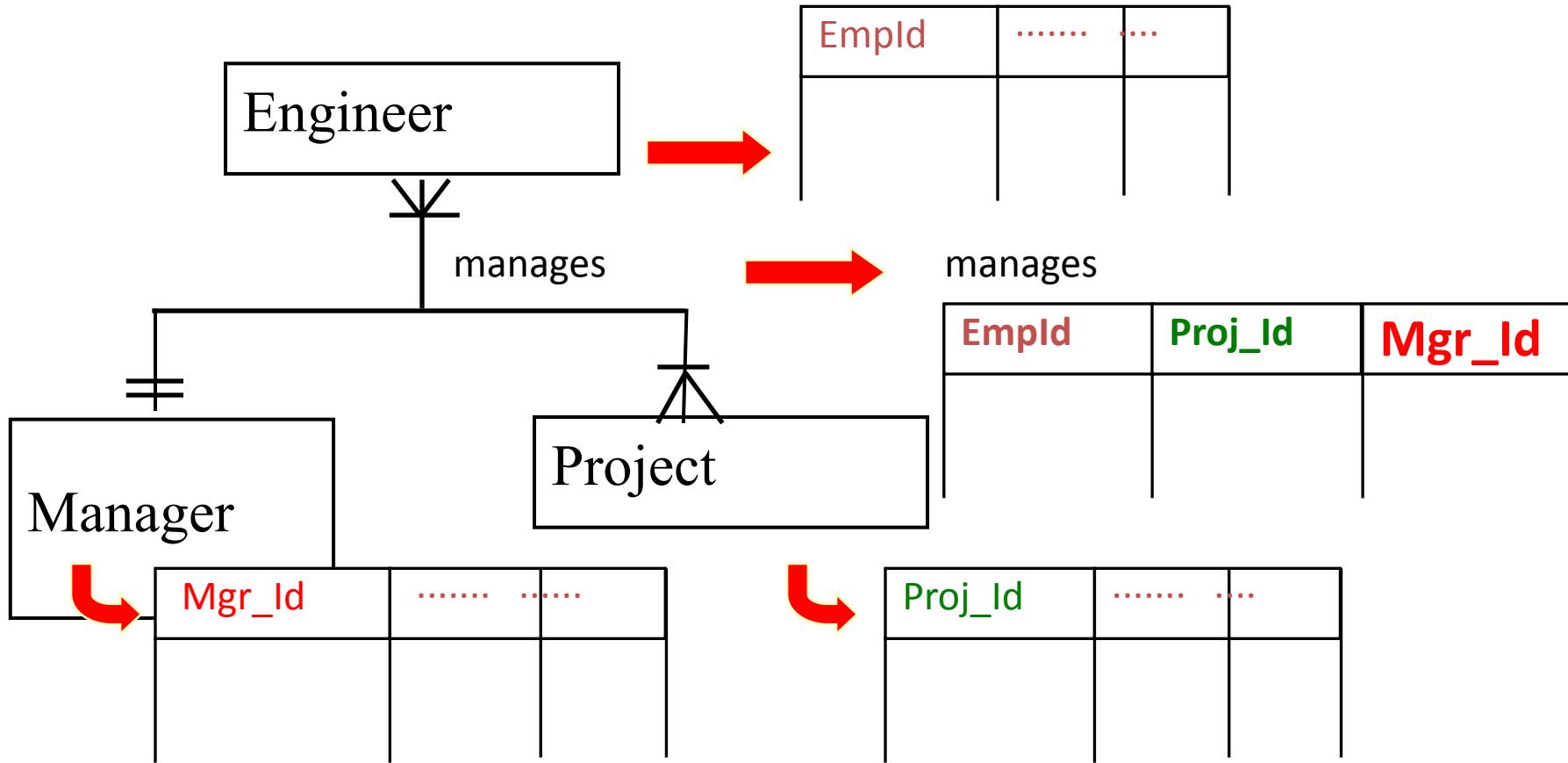
# Primary Many-One- One Mandatory

Each employee assigned to a project works at only one location, but can be at a different location for a different project.



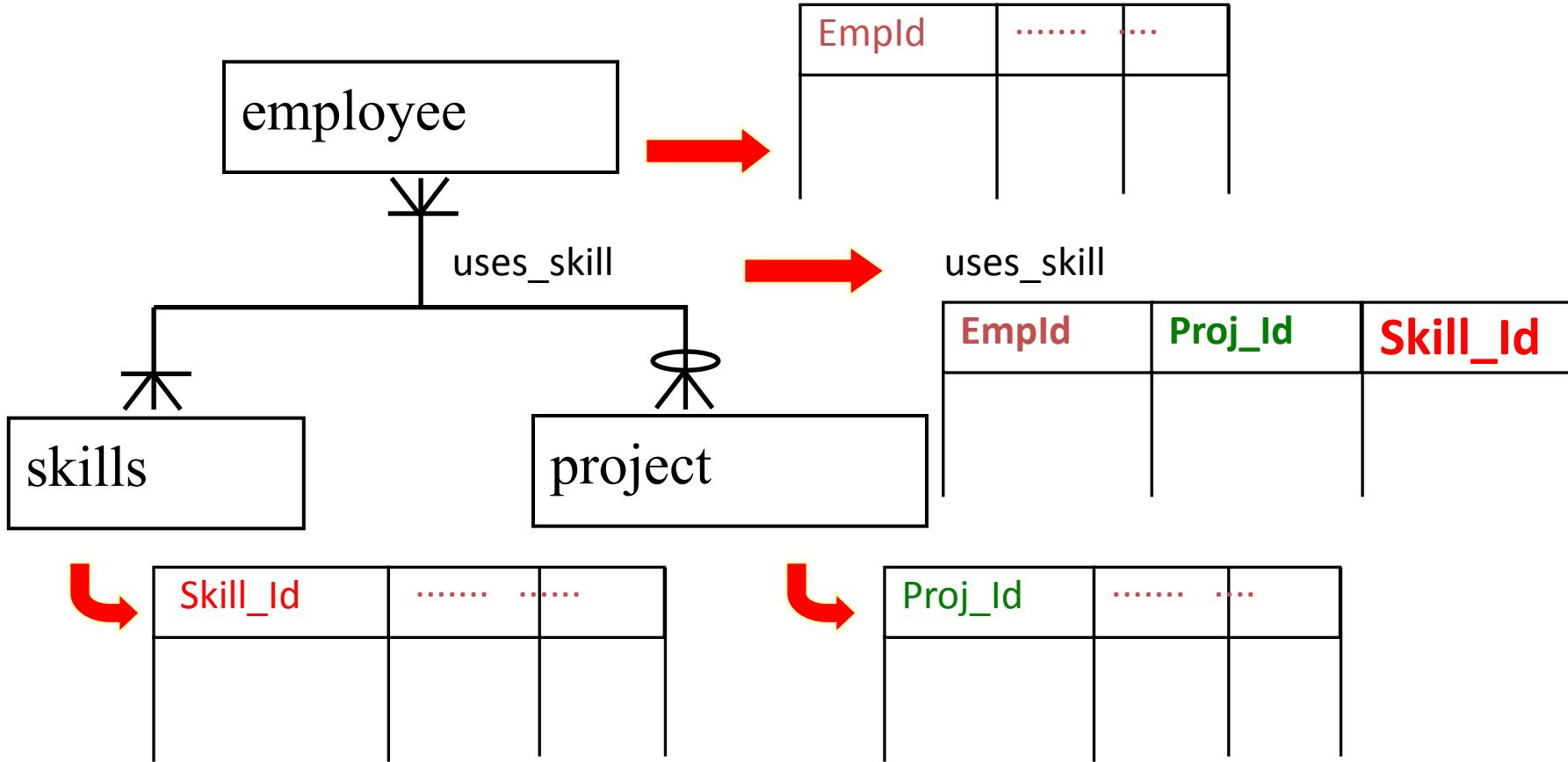
# Ternary Many-Many-One Mandatory

Each engineer working on a particular project has exactly one manager, but a project may have many managers and many projects.



# Ternary Many-Many-Many Optional

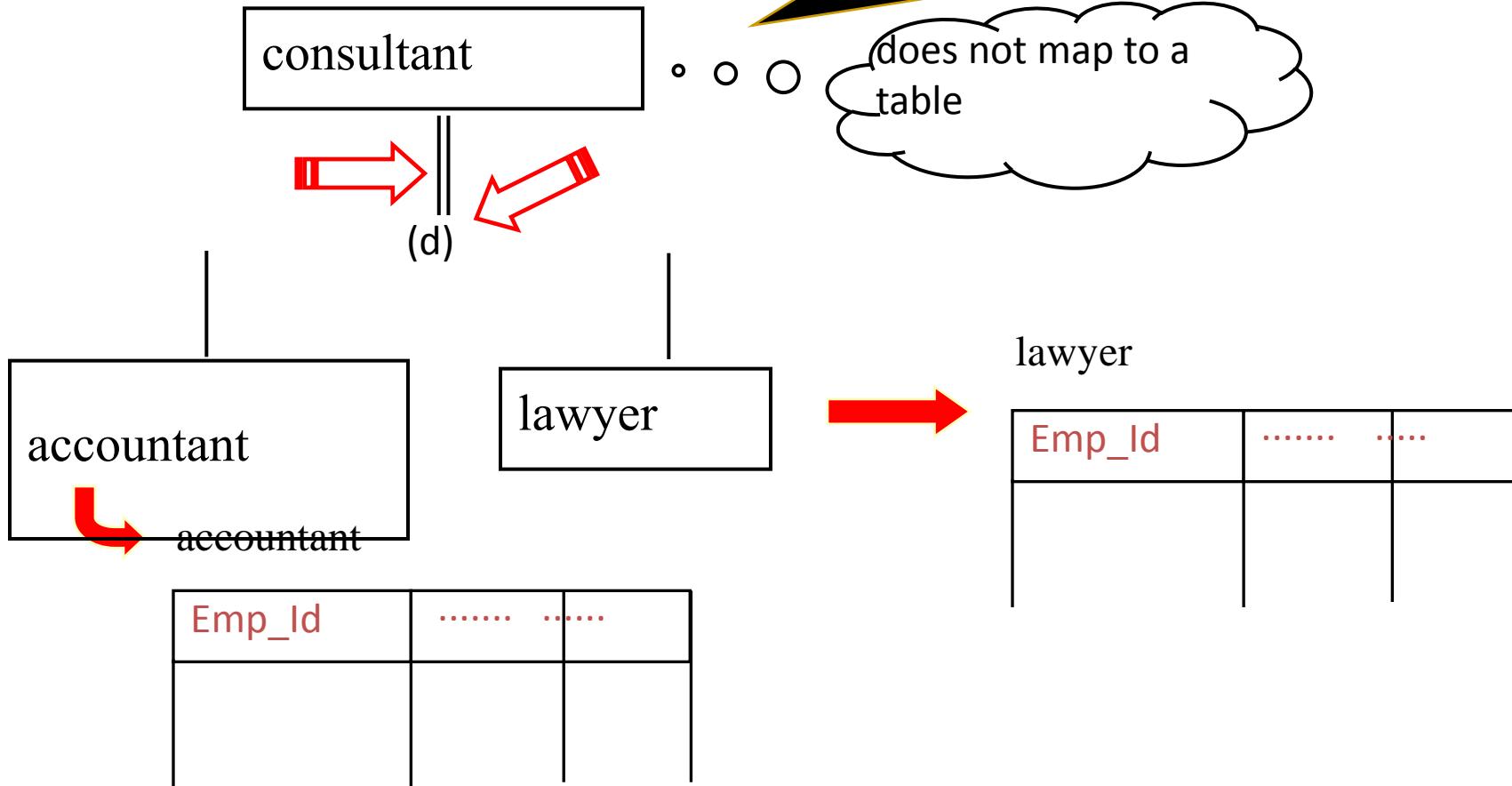
Employees use a wide range of different skills for each project.



# Specialisation Hierarchies

1

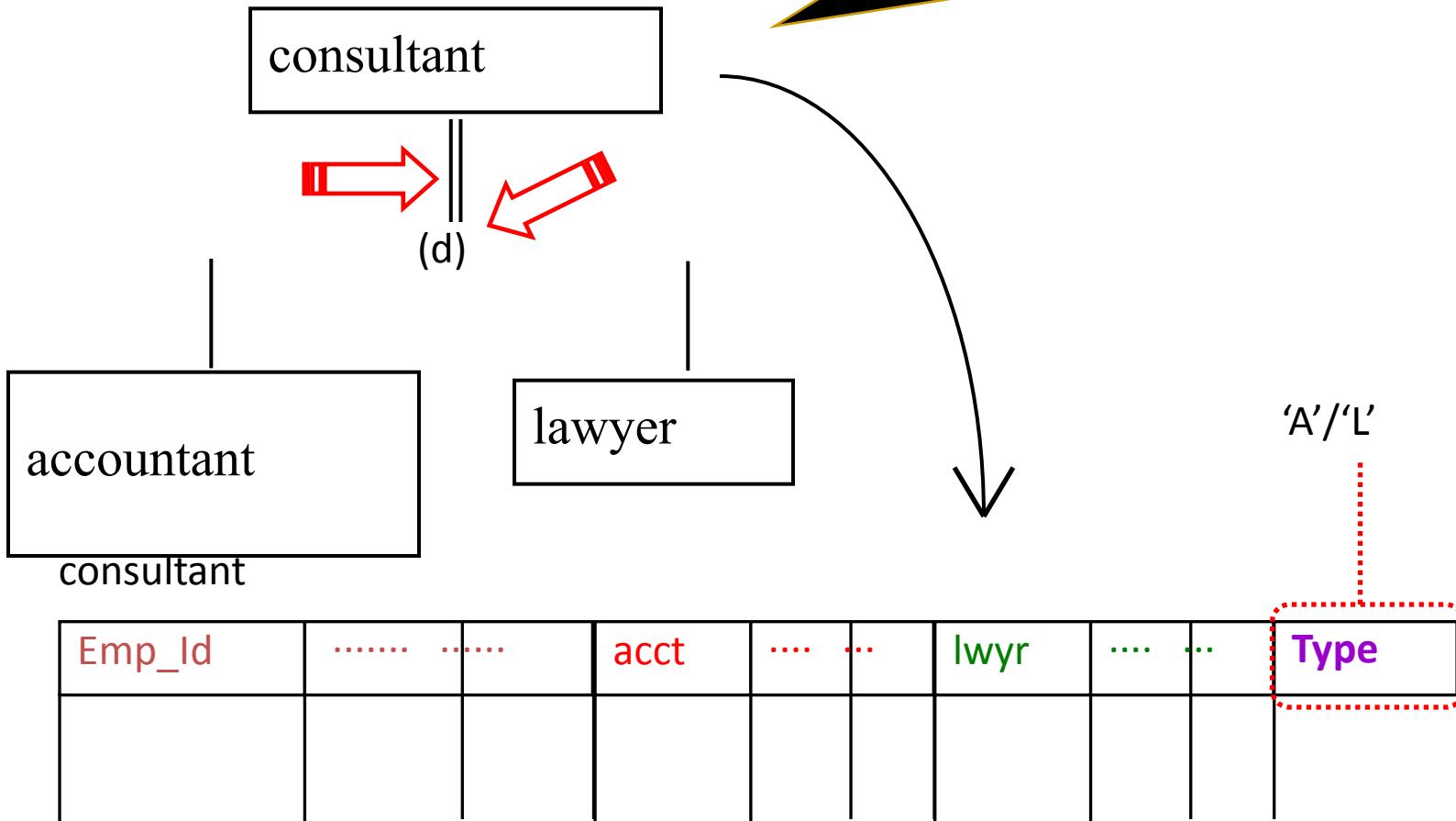
A consultant is either an accountant or a lawyer but not none and not both.



# Specialisation Hierarchies

2

A consultant is either an accountant or a lawyer but not none and not both.



# **Introduction to Relational Model**

# Example of a RELATION

Consider the relation EMPLOYEE represented by the following table:

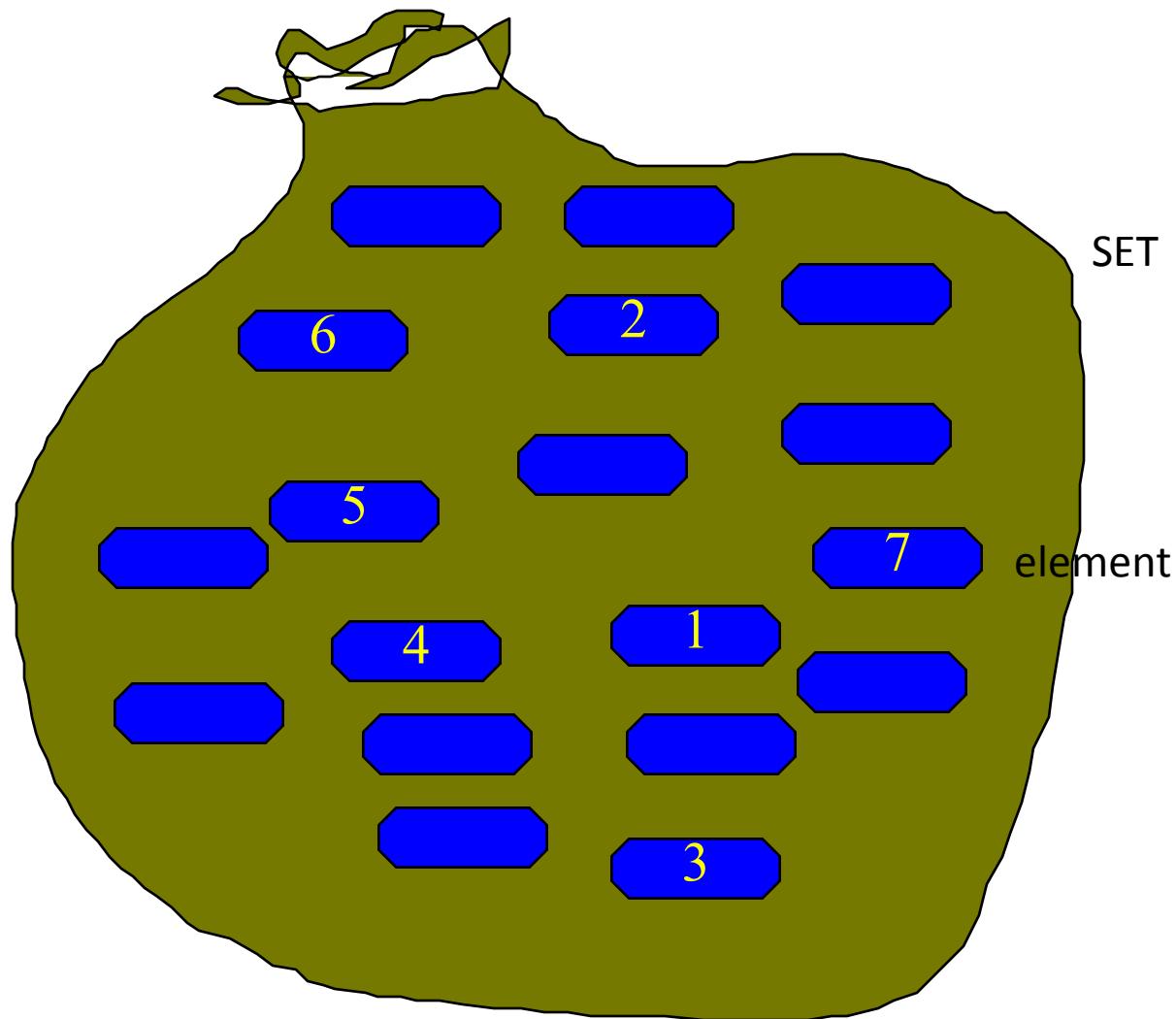
Emp Code	Name	Desig Code	Grade	Join Date	Basic Salary	Sex	Dept Code

# Tuples of a RELATION

Each row here is a tuple.

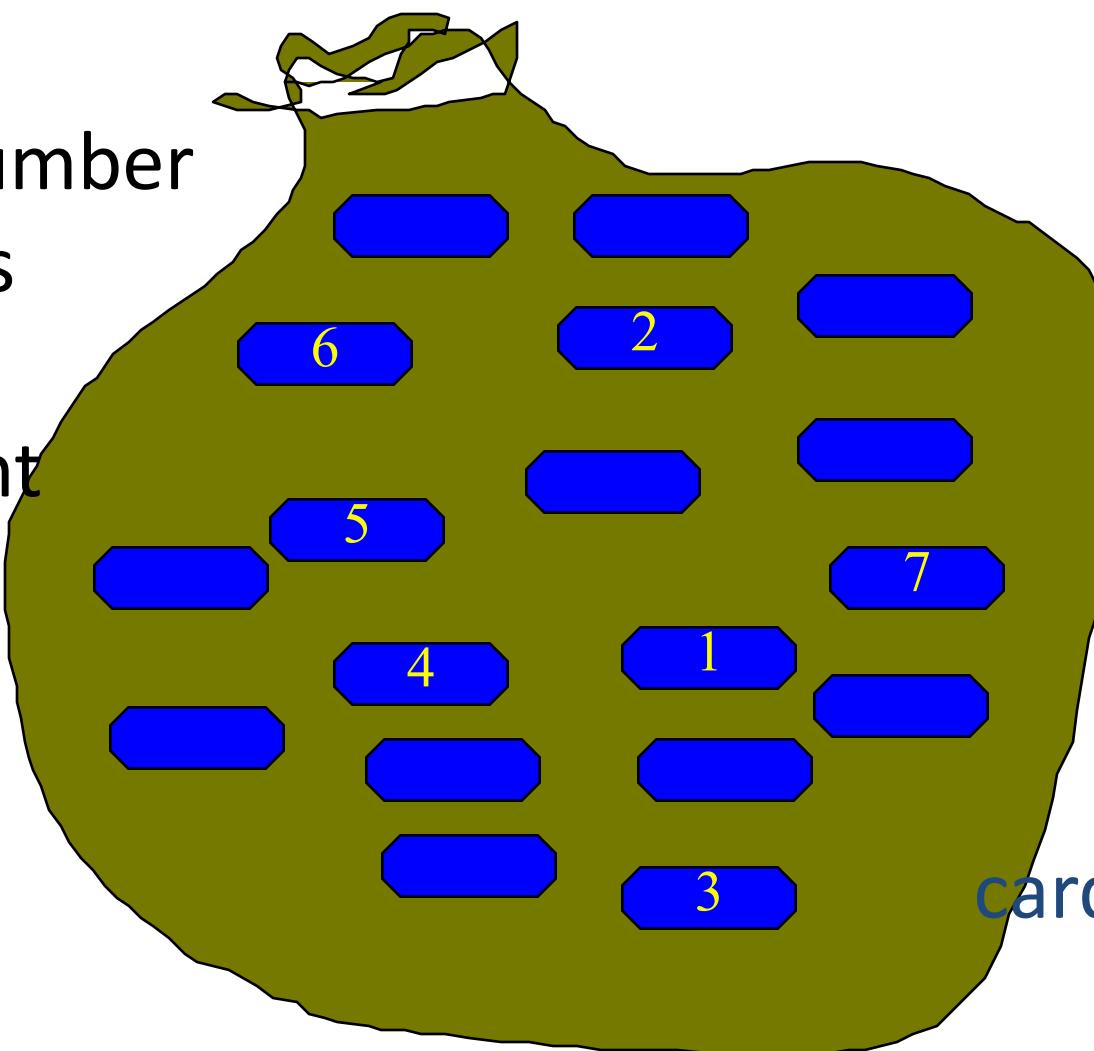
Emp Code	Name	Desig Code	Grade	Join Date	Basic Salary	Sex	Dept Code
							1
							2
							3
							4
							5
							6
							7
							8

# RELATION is a set of Tuples



# Cardinality of a RELATION

is the number  
of tuples  
in it,  
at a point  
in time



cardinality = 18

# Arity (Degree) of a Relation

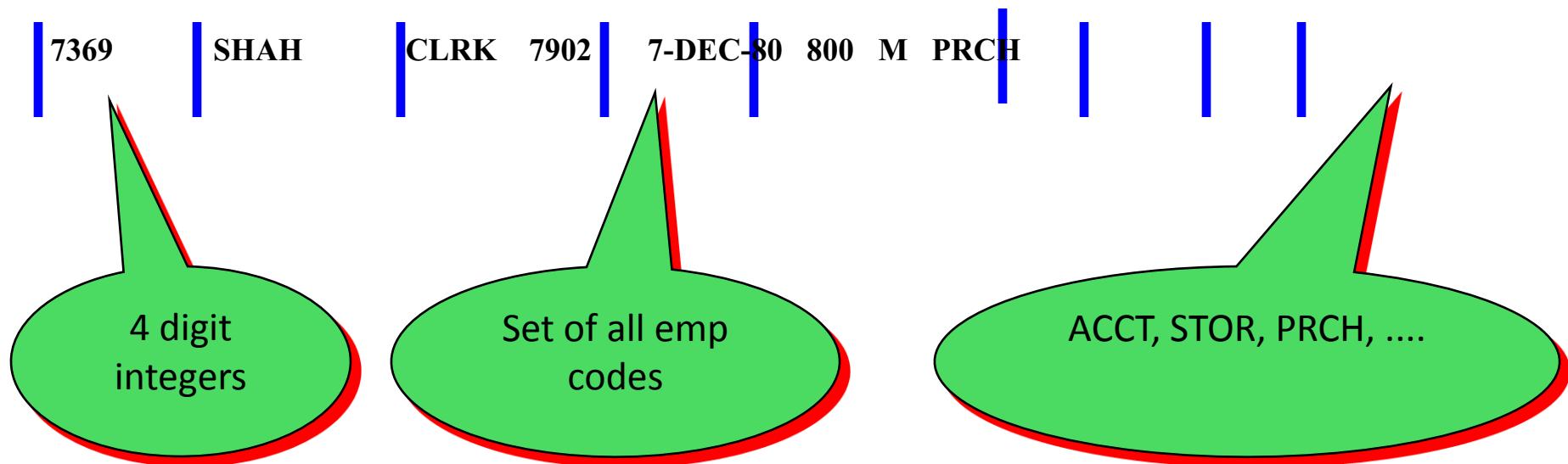
is the number of attributes in it.



$$\text{arity} = 8$$

# Domains

Each attribute has a domain associated with it. Attribute values in a relation are restricted to the values from its domain.



Consider the relation EMPLOYEE defined as:

```
create table EMPLOYEE{  
    EmpCode    integer(4),  
    Name       char(30),  
    DesigCode  char(4),  
    Grade      integer(4),  
    JoinDate   date,  
    Basic      money,  
    Sex        char(1),  
    DeptCode   char(4) }
```

# Domains of attributes of Employee

EmpCode	set of all 4-digit numbers
Name	set of all 30-alpha characters
DesigCode	set of all designation codes
Grade	set of all grade values
JoinDate	set of all dates (in a given range)
Basic	set of all possible values for basic
Sex	set {'M','F'}
DeptCode	set of all dept codes

A relation may be *represented* as a table where

### Relation

Tuple

Attribute

Arity

Primary key

Domain

Cardinality

### Table

Row

Column

Number of columns

Unique identifier

Pool of acceptable values

Number of rows

# Super key

A set of attributes is said to be super key if and only if it satisfies:

- ***Uniqueness property:*** No two distinct tuples have the same value for the key.
- If R is a relation, the set of attributes K is a Super Key, iff, for two distinct tuples  $t_1$  and  $t_2$  in R,  $t_1[K] \neq t_2[K]$

# Candidate key

A set of attributes is said to be candidate key if and only if it satisfies the following time-independent properties:

- ***Uniqueness property: No two distinct tuples have the same value for the key.***
- ***Minimality property: None of the attributes of the key can be discarded from the key without destroying the uniqueness property.***
- ***A super key such that no proper subset is a super key within the relation***

# Candidate key

- A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table
  - The least combination of fields distinguishes a candidate key from a super key.
- A candidate is a subset of a super key
  - Every table must have at least one candidate key but at the same time can have several.

# Candidate key

- It must contain unique values
- It must not contain null values
- It contains the minimum number of fields to ensure uniqueness
- It must uniquely identify each record in the table

## Candidate Keys

StudentId	firstName	lastName	courseld
L0002345	Jim	Black	C002
L0001254	James	Harradine	A004
L0002349	Amanda	Holland	C002
L0001198	Simon	McCloud	S042
L0023487	Peter	Murray	P301
L0018453	Anne	Norris	S042

- `student_id` uniquely identifies the students in a student table. This would be a candidate key.
- student's first name and last name that also, when combined, uniquely identify the student in a student table.
- Both can be candidate keys.

# Primary key

One of the candidate keys is chosen as the Primary Key.

There can be only one primary key per relation.

Primary key may be a compound key.

# Primary Key

```
create table EMPLOYEE{  
    EmpCode    char(6)          primary key,  
    Name       char(30),  
    DesigCode  char(4),  
    GradeCode  integer(4),  
    JoinDate   date,  
    Basic      money,  
    Sex        char(1),  
    DeptCode   char(4) }
```

## Secondary Key or Alternative Key

Any candidate key which is not a Primary Key is an Alternate Key.

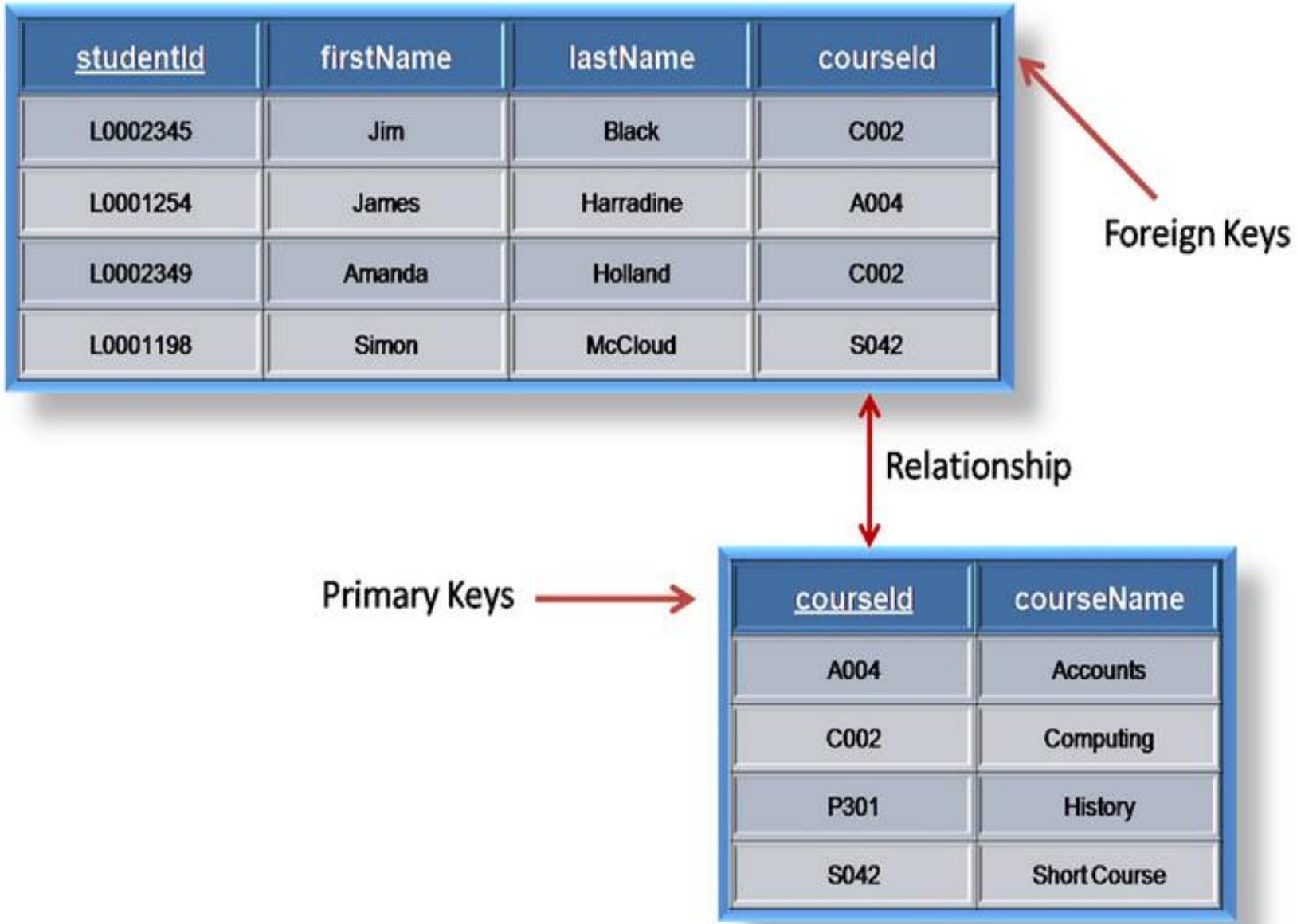
There can be more than one alternate key for any given relation.

# Alternate Key

```
create table EMPLOYEE{  
    EmpCode    char(6),  
    Name       char(30)      alternate key,  
    DesigCode  char(4),  
    GradeCode  integer(4),  
    JoinDate   date,  
    Basic      money,  
    Sex        char(1),  
    DeptCode   char(4) }
```

# Foreign Key

- A foreign key is generally a primary key from one table that appears as a field in another where the first table has a relationship to the second.
- If we had a table A with a primary key X that linked to a table B where X was a field in B, then X would be a foreign key in B.
- There can be more than one foreign key in a given relation.



# Fundamental Integrity Rules

Entity Integrity - No attribute participating in the primary key of a base relation may accept null values.

*Guarantees that each entity will have a unique identity.*

# Referential Integrity

Values of the foreign key (a) must be either null, or (b) if non-null, must match with the primary key value of some tuple of the ‘parent’ relation.

The reference can be to the same relation.

# Codd's 12 rules for RDBMS

# Codd's 12 Rules

Rule(0) –

The system must qualify as relational, as a database and a management system.

Rule(1) – Informational Rule

All information in a relational database (including table and column names) is represented in only one way, namely as a value in a table.

# Codd's 12 Rules

## Rule(2) – Guaranteed Access Rule

All data must be accessible. *Each and every datum (atomic value) is guaranteed to be logically accessible by resorting to a combination of table name, primary key value and column name.*

# Codd's 12 Rules

- Rule(3) – *Systematic treatment of null values*
  - The DBMS must allow each field to remain null (or empty) to represent ‘missing’ or ‘not applicable’ information.
  - NULL can be interpreted as one the following: data is missing, data is not known, data is not applicable etc

# Codd's 12 Rules

- Rule(4) – *Active online catalog based on the relational model*
  - the structure description of whole database must be stored in an online catalog
  - data dictionary, which can be accessed by the authorized users.
  - Users can use the same query language to access the catalog which they use to access the database itself.

# Codd's 12 Rules

## Rule(5) – *Comprehensive Data Sub-language*

The system must support at least one relational language that

- Has a linear syntax
- Can be used both interactively and within application programs,
- Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity constraints, and transaction management operations (begin, commit, and rollback).

# Codd's 12 Rules

Rule(6) – The *view updating rule*

All views that are theoretically updatable must be updatable by the system.

Rule(7) – High-level insert, update, and delete

# Codd's 12 Rules

## Rule(8) – Physical Data Independence

- Application should not have any concern about how the data is physically stored.
- Any change in its physical structure must not have any impact on application.

## Rule(9) – Logical Data Independence

- Logical data must be independent of its user's view (application).
- Any change in logical data must not imply any change in the application using it.
- if two tables are merged or one is split into two different tables, there should be no impact the change on user application.

# Codd's 12 Rules

## Rule(10) – Integrity Independence

- Database must be independent of the application using it.
- All its integrity constraints can be independently modified without the need of any change in the application.
- This rule makes database independent of the front-end application and its interface.

# Codd's 12 Rules

## Rule(11) – Distribution Independence

End user must not be able to see that the data is distributed over various locations.

User must also see that data is located at one site only.

This rule has been proven as a foundation of distributed database systems.

# Codd's 12 Rules

## Rule(12) – Subversion Rule

If a system has an interface that provides access to low level records, this interface then must not be able to subvert the system and bypass security and integrity constraints.

# Codd's Objectives in Developing the Relational Model

- To accomplish a high degree of data independence
- To introduce a theoretical foundation for database management
- To eventually infuse inferential capabilities into a data management system

# Content

- Introduction to SQL
- Categories of SQL Commands: DDL, DML, DCL, DTL/TCL
- DDL (CREATE/ALTER/DROP/TRUNCATE)
- DML (INSERT/UPDATE/DELETE)
- MySQL Data Types
- Database Constraints (Primary Key, Unique, Not Null, Foreign Key, Default, Check\*)
- Aggregate Functions, Grouping Things Together (Group By, Having)
- LIKE Operator, DISTINCT, Sorting (Order by clause)
- BETWEEN AND Operators, Comparing Nulls (IS NULL/IS Not NULL), IN/NOT IN

# What is SQL ?

- ▶ *Structured Query Language.*
- ▶ Access mechanism to a relational database and therefore at the heart of any contemporary RDBMS.

# Components of SQL:

- Data Definition Component  
*a.k.a. Data Definition Language (DDL)*

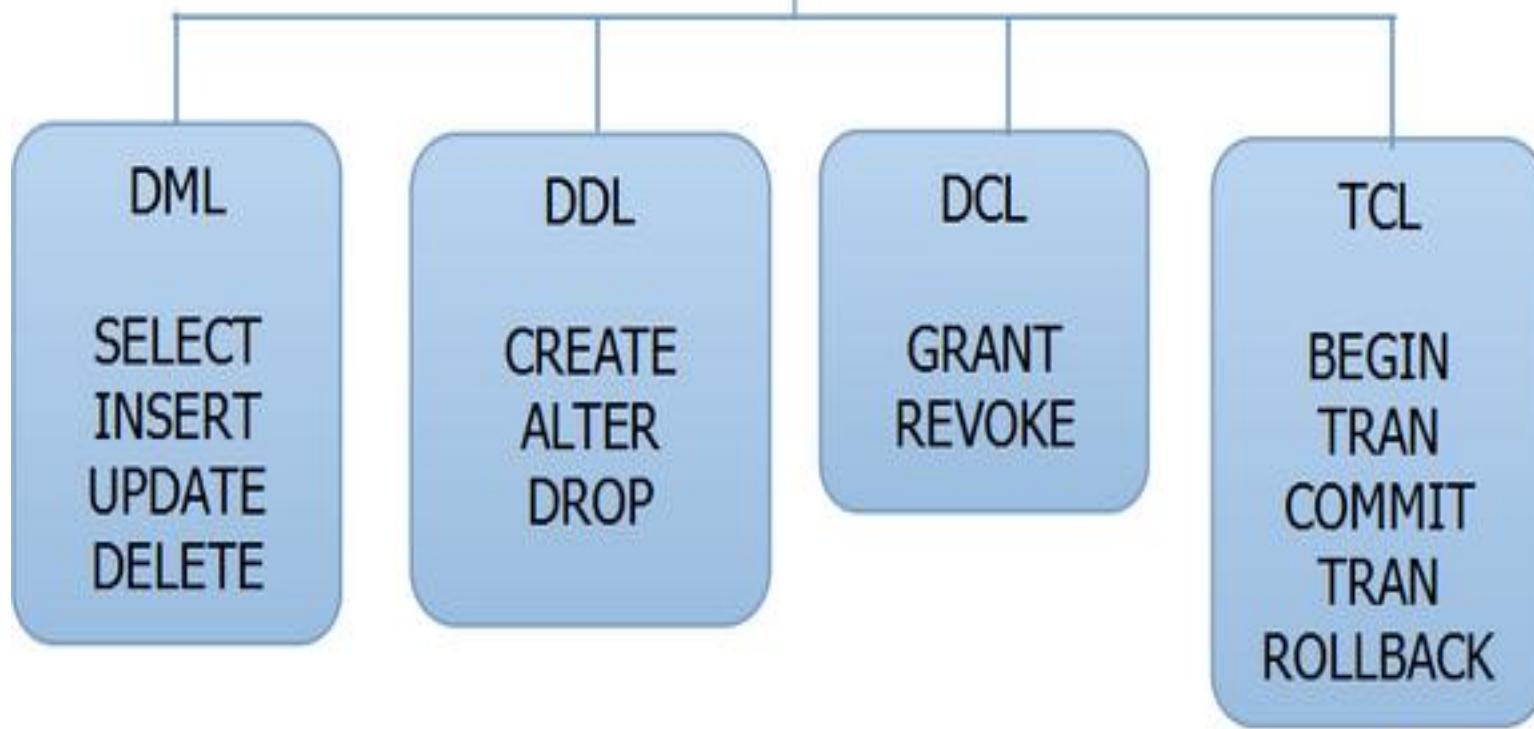
---
- Data Manipulation Component  
*a.k.a. Data Manipulation Language (DML)*

---
- Transaction Control Component
- Others  
*View Definition, Embedded DML, Integrity*

# SQL in action:

- Single table queries
- Multiple table queries
- Aliases
- Subqueries
- Aggregate functions
- Advanced SQL features

## SQL Language Statements



# Data Definition Language (DDL) Statements

- The CREATE, ALTER, and DROP commands require exclusive access to the specified object.
  - ALTER TABLE statement fails if another user has an open transaction on the specified table.
- The GRANT, REVOKE, ANALYZE, AUDIT, and COMMENT commands do not require exclusive access to the specified object.
  - You can analyze a table while other users are updating the table.

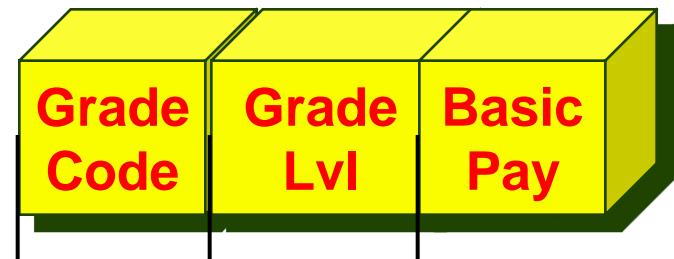
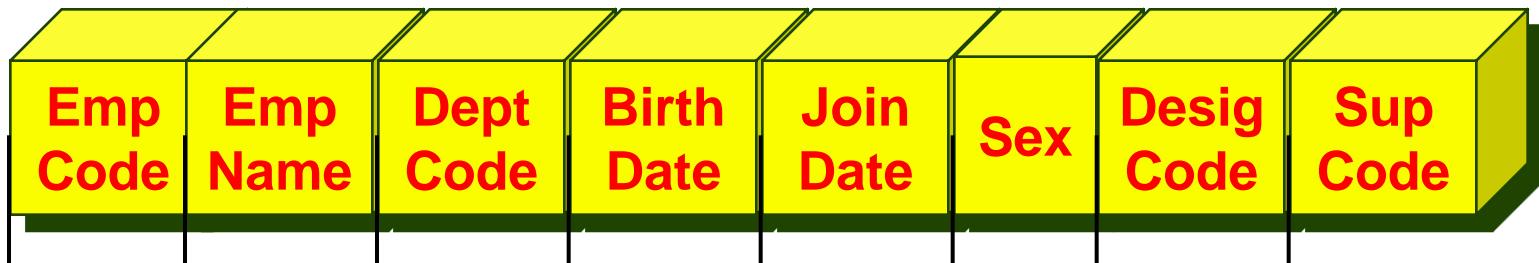
# Data Manipulation Language (DML) Statements

- Data manipulation language (DML) statements access and manipulate data in existing schema objects.
- These statements do not implicitly commit the current transaction.
- The SELECT statement is a limited form of DML statement in that it can only access data in the database

# **Transaction control statements**

- Transaction control statements manage changes made by DML statements.

# Emp Table



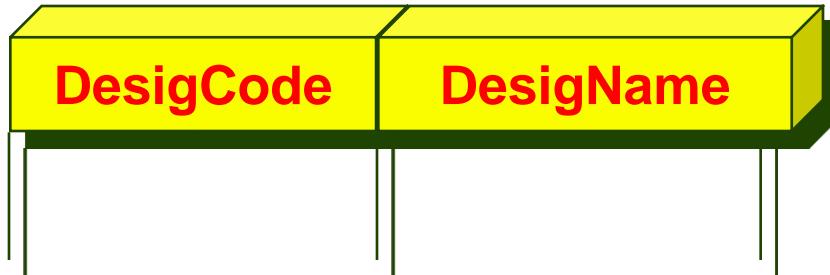
# Dept Table

DeptCode	DeptName	DeptManager	DeptBudget

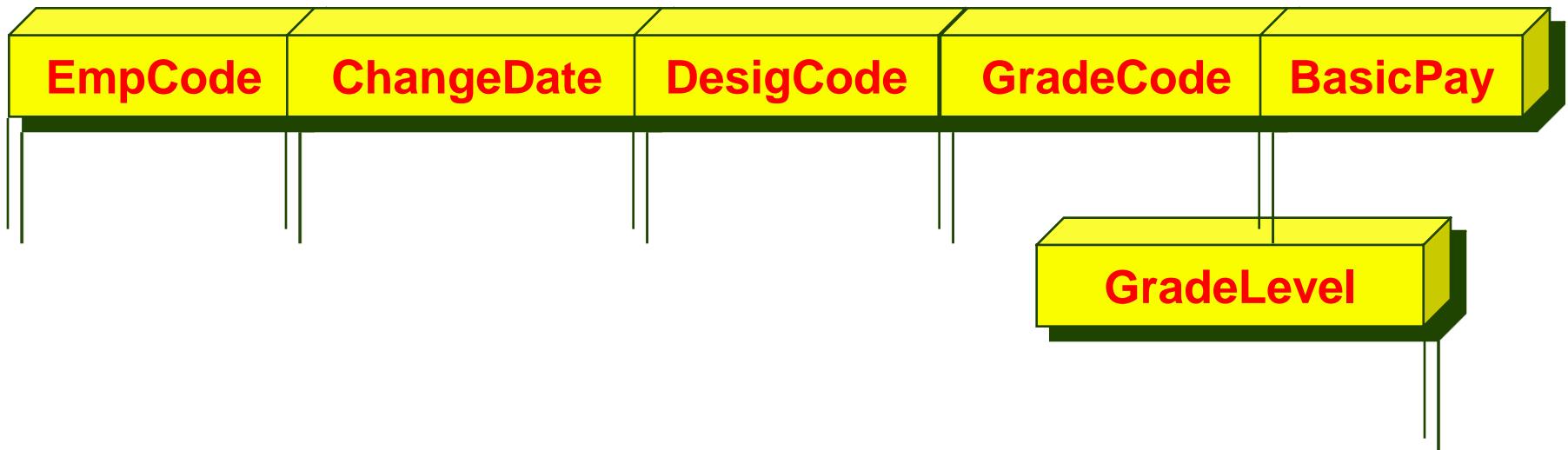
# Salary Table

EmpCode	SalMonth	Basic	Allow	Deduct

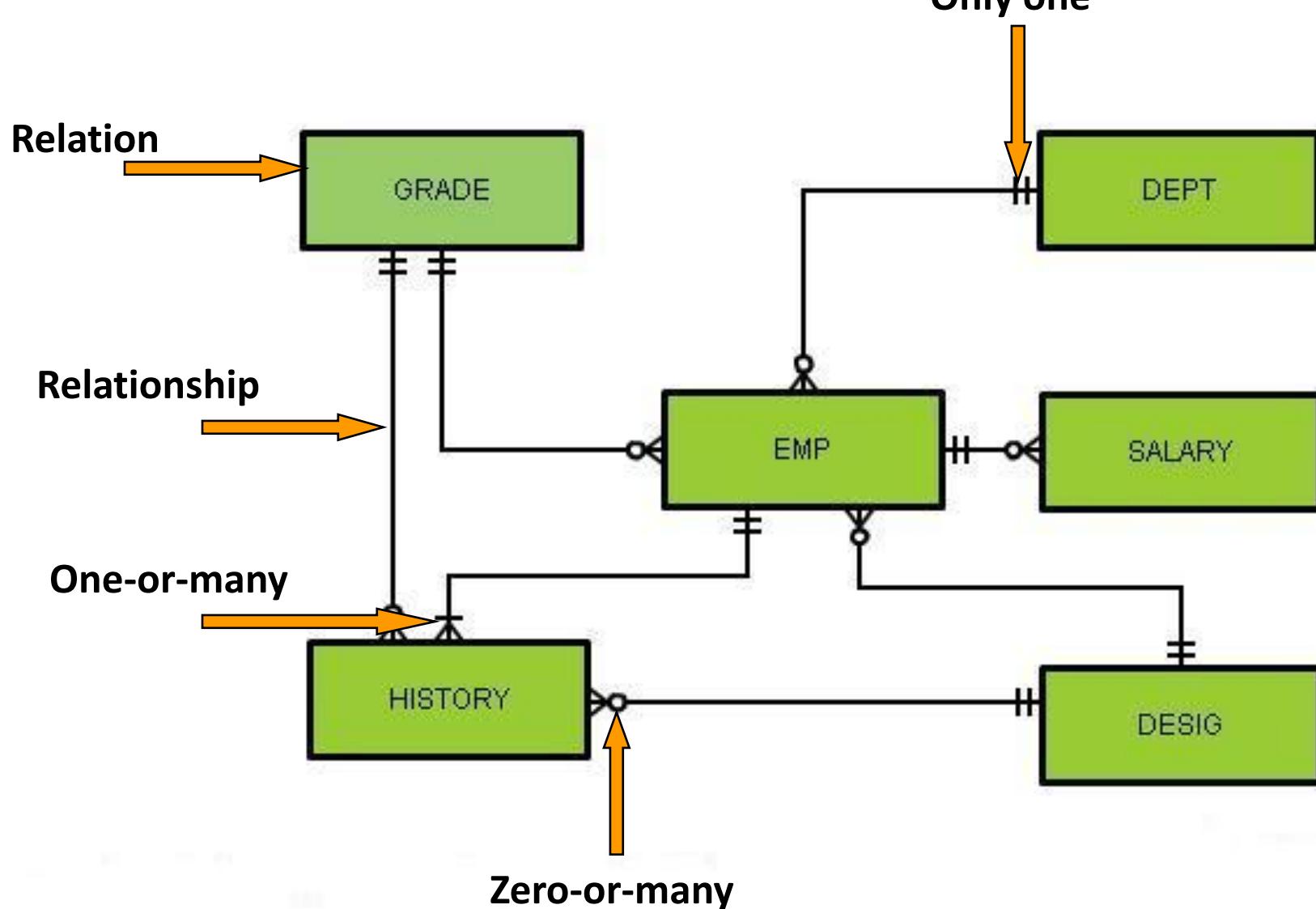
# Desig Table



# History Table



# Schema – FR Diagram



# The Whole Table

1

List the Department table

```
SELECT *
FROM Dept;
```

DEPT	DEPTNAME	DEPTMA	DEPTBUDGET
ACCT	Accounts	7839	19
PRCH	Purchase	7902	25
SALE	Sales	7698	39
STOR	Stores	7521	33
FACL	Facilities	7233	42
PERS	Personnel	7233	12

6 rows selected.

# Another Way - Better!

**2**

**List the Department table**

```
SELECT DeptCode, DeptName,  
       DeptManager, DeptBudget  
  
FROM      Dept;
```

DEPT	DEPTNAME	DEPTMA	DEPTBUDGET
ACCT	Accounts	7839	19
PRCH	Purchase	7902	25
SALE	Sales	7698	39
STOR	Stores	7521	33
FACL	Facilities	7233	42
PERS	Personnel	7233	12

6 rows selected.

# Only Some Columns

3

List all department managers with the names of their departments

```
SELECT DeptManager, DeptName  
FROM      Dept;
```

```
DEPTMA DEPTNAME
```

```
-----  
7839    Accounts  
7902    Purchase  
7698    Sales  
7521    Stores  
7233    Facilities  
7233    Personnel
```

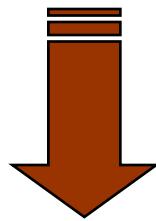
```
6 rows selected.
```

# SQL query structure:

**SELECT** A<sub>1</sub>, A<sub>2</sub>, ..., A<sub>n</sub>

**FROM** r<sub>1</sub>, r<sub>2</sub>, ..., r<sub>m</sub>

**WHERE** P;



$\Pi_{A_1, A_2, \dots, A_n} (\sigma_P (r_1 \times r_2 \times \dots \times r_m))$

# DISTINCT

4

List all department managers

```
SELECT DeptManager  
FROM Dept;
```

duplicates are  
NOT eliminated !

DEPT
----
Khan
Khan
Roy
Patil
Khan

# DISTINCT

5

List all department managers

```
SELECT      DISTINCT DeptName  
FROM        Dept;
```

DEPT
-----
Khan
Roy
Patil

# WHERE Predicate (=)

6

List all employees of the Accounts department

```
SELECT          EmpName, DeptCode  
  FROM          Emp  
 WHERE         DeptCode = 'ACCT';
```

EMPNAME	DEPT
-----	-----
Reddy	ACCT
Menon	ACCT
Kaul	ACCT

# WHERE Predicate (=)

7

List all officers

```
SELECT EmpName, GradeCode  
FROM   Emp  
WHERE  GradeCode = 'GC6';
```

EMPNAME	GRADECODE
Naik	4
Reddy	1
Murthy	4
Wilson	4
Jain	4
Menon	4
Khan	6
Kumaran	4
Kamal	4

9 rows selected.

# SELECT from more than a table

8

List the employees with their department name

```
SELECT EmpName, DeptName  
FROM Emp, Dept  
WHERE Emp.DeptCode = Dept.DeptCode;
```

# SELECT from more than a table

8

List the employees with their department code and department name

```
SELECT EmpName, DeptName, Dept.DeptCode  
FROM Emp, Dept  
WHERE Emp.DeptCode = Dept.DeptCode;
```

EMPNAME	DEPTOCDE	DEPTNAME
Shah	PRCH	Purchase
Naik	PRCH	Purchase
Reddy	ACCT	Accounts
Jain	PRCH	Purchase
Menon	ACCT	Accounts
.	.	.
.	.	.
.	.	.

17 rows selected.

# Date Comparison

9

List all young employees

```
select empname,birthdate  
from emp  
where birthdate > '1980-01-01';
```

# Set Membership

**10**

List employees of admin departments

```
SELECT EmpName, DeptCode  
FROM Emp  
WHERE DeptCode IN ('ACCT' , 'PRCH' , 'PERS');
```

EMPNAME	DEPT
Shah	PRCH
Naik	PRCH
Reddy	ACCT
Jain	PRCH
Menon	ACCT
Khan	PRCH
Patil	PRCH
Kaul	ACCT
Uma	PERS

9 rows selected.

# Set Membership

11

List employees of non-admin departments

```
SELECT EmpName, DeptCode  
FROM   Emp  
WHERE  DeptCode NOT IN ('ACCT' , 'PRCH' , 'PERS' );
```

# Between Construct

12

List staff between certain age

```
select empname,birthdate  
from emp  
where birthdate between '1982-12-31' and '1992-12-31';
```

EMPNAME	BIRTHDATE
Rai	08/10/1988
Tiwari	08/19/1989

# Like Construct

13

List all employees with UMA in their names

```
SELECT EmpName  
FROM Emp  
WHERE Upper(EmpName) LIKE '%UMA%';
```

EMPNAME

-----

Kumaran

Uma

# Null Values

14

List the employees who have not been assigned to any supervisor

```
SELECT EmpName, SupCode  
FROM Emp  
WHERE SupCode IS NULL
```

EMPNAME	SUPCOD
Reddy	

# Compound Predicate - Interval

15

List the female employees who have just completed 5 years

```
SELECT EmpName, Sex, JoinDate  
FROM Emp  
WHERE Sex = 'F' AND AND  
      JoinDate BETWEEN (CURRENT_DATE - 5*365)  
                    AND (CURRENT_DATE - 6*365);
```

EMPNAME	S	JOINDATE
Uma	F	22-OCT-91

# Predicate using OR

16

List FEMALE employees who are either 50 years or more or have more than 20 years experience

```
SELECT EmpName  
FROM Emp  
WHERE BirthDate < (CURRENT_DATE - 50*365)  
OR JoinDate < (CURRENT_DATE - 20*365) and SEX= 'F';
```

EMPNAME
-----
Reddy
Kumaran
Kamal

# Parentheses in Predicate

17

List salesmen who are either 50 years or more  
or have more than 20 years' experience

```
SELECT EmpName
FROM   Emp
WHERE  DesigCode = 'SLMN'
AND    (BirthDate < (CURRENT_DATE - 50*365)
          OR
          JoinDate < (CURRENT_DATE - 20*365));
```

EMPNAME
Kumaran
Kamal

# Expressions in SELECT List

**18**

**List 1% of take-home pay of all employees**

```
SELECT EmpCode, (Basic + Allow - Deduct) * 0.01  
FROM Salary  
WHERE SalMonth = '02/01/2012';
```

EMPCOD (BASIC+ALLOW-DEDUCT) \*0.01

7129	440
7233	440
7345	143
7369	66
.	.
.	.
.	.
7844	198
7876	44
7900	55
7902	330
7934	77

17 rows selected.

# Aliasing

19

List the present age of all the employees

```
select empname,timestampdiff(year,birthdate,curdate()) as age  
from emp;
```

# Defining a Column Alias

👉 A column alias:

- Renames a column heading
- Is useful with calculations
- Immediately follows the column name - there can also be the optional AS keyword between the column name and alias

EMPNAME	AGE
Shah	20
Naik	36
Reddy	54
Murthy	34
Roy	24
.	.
.	.
.	.
Shroff	19
Kaul	21
Kumaran	57
Kamal	46
Uma	22

17 rows selected.

# ALL construct

20

List the employees with the highest salary

```
SELECT EmpName, basicpay  
FROM   Emp  
WHERE  BasicPay >= ALL( SELECT BasicPay  
                           FROM   Emp);
```

EMPNAME	BasicPay
Reddy	40000

- ☞ The ALL,ANY comparison condition is used to compare a value to a list or subquery.
- ☞ It must be preceded by =, !=, >, <, <=, >= and followed by a list or subquery.
- ☞ When the ALL condition is followed by a list, the optimizer expands the initial condition to all elements of the list and strings them together with AND operators
- ☞ When the ANY condition is followed by a list, the optimizer expands the initial condition to all elements of the list and strings them together with OR operators

## ALL

- $\text{Page} > \text{ALL}(4,2,7)$  – Page is greater than items in the list (4,2,7) – anything larger than 7 qualifies.
- $\text{Page} < \text{ALL}(4,2,7)$  – Page less than lowest items in the list (4,2,7) – anything less than 2 qualifies.
- $\text{Page} \neq \text{ALL}(4,2,7)$  – Page not equal to any items in the list (4,2,7) – any number qualifies except 4,2 and 7.

## ANY

- $\text{Page} > \text{ANY}(4,2,7)$  – Page is greater than any single items in the list (4,2,7)
  - 3 qualifies, because it is greater than 2
- $\text{Page} < \text{ANY}(4,2,7)$  – Page is less than any single items in the list (4,2,7)
  - 6 qualifies as it is less than 7.
- $\text{Page} \neq \text{ANY}(4,2,7)$  – Page not equal to any single item in the list (4,2,7)

# ANY construct

21

List all the employees not having the highest salary

```
SELECT EmpName  
FROM Emp  
WHERE BasicPay < ANY (SELECT BasicPay  
                      FROM Emp);
```

# UNION and UNION ALL

**22**

List the employees working for ‘accounts’ or  
‘purchase’ departments

```
SELECT EmpName
FROM   Emp
WHERE  DeptCode = 'ACCT'
      UNION
SELECT EmpName
FROM   Emp
WHERE  DeptCode = 'PRCH';
```

To show duplicates, use UNION ALL

EMPNAME	DEPT
Shah	PRCH
Naik	PRCH
Reddy	ACCT
Jain	PRCH
Menon	ACCT
Khan	PRCH
Patil	PRCH
Kaul	ACCT

8 rows selected.

## Restriction on UNION, INTERSECT and MINUS

- Queries that use UNION, INTERSECT and MINUS in where clause must have the same number and type(data type) of columns in their select list
- Only the column names from the first select statement can be used in **order by** clause.
- **IN** construction does not have this limitation

# ORDER BY

Emp Name	Grade Code
WILSON	3
JAIN	3
MURTHY	2
MENON	2
KHAN	4
REDDY	1
NAIK	2

No particular order

# ORDER BY

SELECT  
FROM

.....

.....

ORDER BY GradeCode;

No ordering of  
EmpName

Emp Name	Grade Code
REDDY	1
MURTHY	2
MENON	2
NAIK	2
WILSON	3
JAIN	3
KHAN	4

# ORDER BY

**SELECT  
FROM**

.....  
.....  
.....  
.....

**ORDER BY GradeCode, EmpName;**

<b>Emp Name</b>	<b>Grade Code</b>
REDDY	1
MENON	2
MURTHY	2
NAIK	2
JAIN	3
WILSON	3
KHAN	4

# Order by

25

List all employees ordered by age

```
SELECT      EmpName,  
           TRUNC((CURRENT_DATE - BirthDate) / 365) as AGE  
FROM        Emp  
ORDER BY    BirthDate;
```

```
select empname,timestampdiff(year,birthdate,curdate()) as age  
from emp  
order by birthdate;
```

EMPNAME	AGE
Kumaran	57
Reddy	54
Kamal	46
Menon	38
.	.
.	.
.	.
Uma	22
Patil	21
Kaul	21
Shah	20
Shroff	19

17 rows selected.

# Sorting Ascending/Descending

26

List middle level staff according to seniority

```
SELECT      EmpName, GradeCode, GradeLevel  
FROM        Emp  
WHERE       GradeCode BETWEEN 10 AND 25  
ORDER BY    GradeCode, GradeLevel DESC;
```

EMPNAME	GRADECODE	GRADELEVEL
Gupta	12	3
Roy	12	2
Singh	12	1
Uma	15	2
Patil	20	4
Shroff	20	3
Shah	20	2
Kaul	20	1

8 rows selected.

# Aggregate Functions

28

**Count employees reporting to Singh**

```
SELECT COUNT(*)  
FROM Emp  
WHERE SupCode = '7839';
```

COUNT ( \* )

-----

1

# Aggregate Functions

29

Count employees reporting to Singh

```
SELECT COUNT(*)  
FROM Emp  
WHERE SupCode = 'empcode of Singh';
```

# Aggregate Functions

30

Count employees reporting to Singh

```
SELECT COUNT(*)  
FROM Emp  
WHERE SupCode = (SELECT EmpCode  
                  FROM EMP  
                  WHERE EmpName = 'Singh');
```

COUNT ( \* )

1

# Group By - Aggregate Functions

31

List the number of staff reporting to each supervisor

```
SELECT      SupCode, COUNT(*)  
FROM        Emp  
GROUP BY    SupCode  
ORDER BY    SupCode;
```

SUPCOD	COUNT (*)
7566	1
7698	5
7782	1
7788	1
7839	6
7844	1
7902	1
	1

8 rows selected.

# List the total take-home pay during 96-97 for all employees

- select empcode, sum(basic+allow-deduct) as pay from salary where salmonth between '2011-01-12' and '2012-01-12'  
group by empcode  
order by empcode;

EMPCOD	PAY
7129	132000
7233	132000
7345	42900
7369	19800
7499	56100
.	
.	
.	
7876	13200
7900	16500
7902	99000
7934	15400

17 rows selected.

# Group By - Max & Min

33

List the maximum & minimum salaries  
in grades

```
SELECT      GradeCode, MAX(Basic), MIN(Basic)
FROM        Grade
GROUP BY    GradeCode
ORDER BY    GradeCode;
```

GRADECODE	MAX (BASIC)	MIN (BASIC)
1	25000	25000
4	21000	15000
6	13000	11000
12	9000	8000
15	7000	6000
20	3500	2000

6 rows selected.

# Having

**Having clause is similar to where clause, except it's logic is only related to result of group function, as opposed to columns or individual rows, which can still be selected by a where clause.**

# Having

34

List the number of staff reporting to each supervisor having more than 3 people working under them

```
SELECT      SupCode, COUNT(*)  
FROM        Emp  
GROUP BY    SupCode  
HAVING      COUNT(*) > 3  
ORDER BY    SupCode;
```

SUPCOD	COUNT ( * )
7698	5
7839	6

# Where - Group By - Having

35

List the total take-home pay during 2011-2012 for all employees getting a total take-home-pay < Rs. 20000

```
SELECT      EmpCode, SUM( Basic + Allow - Deduct)  
as PAY  
FROM        Salary  
WHERE       SalMonth BETWEEN '12/01/2011'  
           AND '12/01/2012'  
GROUP BY    EmpCode  
HAVING      SUM( Basic + Allow - Deduct) < 40000  
ORDER BY    EmpCode;
```

# Where - Group By - Having

EMPCODE	PAY
6569	38180
7192	37760
7369	37490
7521	37440
7566	38380
7654	38720
7782	38080
7788	36530
7802	38280
7876	37760
7900	37000
7902	36210
7934	36890
7939	36260

# Where - Group By - Having

**37**

List the maximum and minimum basic salary  
in each grade for grades with start < Rs. 4000

```
SELECT GradeCode, MAX(Basic), MIN(Basic)
FROM      Grade
GROUP BY  GradeCode
HAVING   MIN(Basic) < 4000
ORDER BY  GradeCode;
```

GRADECODE	MAX (BASIC)	MIN (BASIC)
20	3500	2000

# Order of execution

1. Choose rows based on where clause
2. Group rows together based on group by clause
3. Calculate result of group functions for each group
4. Choose and eliminate groups based on having clause
5. Order the group based on result of group function in the Order by clause.
7. The order by clause must use either a group function or a column specified in the group by clause.

Order of execution has impact on performance of the query.  
a.If more record can be eliminated by where clause faster the query will perform, because less number of rows will be Processed in group by clause.

# INNER JOIN

40

List employees along with the names of their supervisors

```
SELECT E.EmpCode, E.EmpName,  
       S.EmpCode, S.EmpName
```

```
FROM   Emp E INNER JOIN Emp S  
ON     E.SupCode = S.EmpCode;
```

EMPCOD	EMPNAME	EMPCOD	EMPNAME
7369	Shah	7902	Naik
7902	Naik	7839	Reddy
7698	Murthy	7839	Reddy
7499	Roy	7698	Murthy
7521	Wilson	7698	Murthy
.			
.			
.			
7900	Shroff	7698	Murthy
7934	Kaul	7782	Menon
7233	Kumaran	7839	Reddy
7129	Kamal	7839	Reddy
7345	Uma	7844	Singh

16 rows selected.

# INNER JOIN :

38

List employees along with their basic salary

```
SELECT EmpCode, EmpName, Salary.Basic  
FROM Emp INNER JOIN Salary  
USING (EmpCode);
```

Inner joins are default, it return rows of two table have in common.  
On/ Using- To specify join criteria

# NATURAL JOIN :

**39**

**List employees along with their basic salary**

```
SELECT EmpCode, EmpName, Basic  
FROM    Emp NATURAL JOIN Salary ;
```

Natural – Join should be performed based on all columns that have the same names, in the two tables being joined

EMPCOD	EMPNAME	BASIC
7369	Shah	3000
7902	Naik	15000
7839	Reddy	25000
7698	Murthy	17000
7499	Roy	8500
.		
.		
.		
7934	Kaul	3500
7233	Kumaran	20000
7129	Kamal	20000
7345	Uma	6500

17 rows selected.

# INNER JOIN (Self Join):

41

List employees along with the names of their supervisors

```
SELECT E.EmpCode, E.EmpName,  
       S.EmpCode, S.EmpName
```

```
FROM   Emp E INNER JOIN Emp S  
ON     E.SupCode = S.EmpCode;
```

# INNER JOIN : Where clause

42

List employees along with the names of their department for which they are working

```
SELECT E.EmpCode, E.EmpName, D.DeptName
```

```
FROM Emp E, Dept D
```

```
WHERE E.DeptCode = D.DeptCode;
```

EMPCOD	EMPNAME	DEPTCODE
7369	Shah	PRCH
7902	Naik	PRCH
7698	Murthy	SALE
7499	Roy	SALE
7521	Wilson	STOR
.		
.		
.		
7900	Shroff	SALE
7129	Jain	PRCH

20 rows selected.

# RIGHT Outer JOIN :

43

List employees along with the names of their department for which they are working. The list should have all the departments listed.

```
SELECT E.EmpName, D. DeptCode  
FROM   Emp E RIGHT OUTER JOIN Dept D  
ON     E.DeptCode = D.DeptCode;
```

# RIGHT Outer JOIN :

43

List employees along with the names of their department for which they are working. The list should have all the departments listed.

```
SELECT E.EmpName, D. DeptCode  
FROM   Emp E RIGHT OUTER JOIN Dept D  
USING (DeptCode);
```

EMPCOD	EMPNAME	DEPTCODE
7369	Shah	PRCH
7902	Naik	PRCH
7698	Murthy	SALE
7499	Roy	SALE
7521	Wilson	STOR
.		
.		
.		
7900	Shroff	SALE
7129	Jain	PRCH
-----	-----	PERS
-----	-----	RCMT
-----	-----	FACL

23 rows selected.

# Full Outer JOIN :

44

```
SELECT E.EmpName, E.DeptCode, D.DeptCode  
FROM   Emp E FULL OUTER JOIN Dept D  
ON     E.DeptCode = D.DeptCode;
```

**46**

**List the number of officers reporting to each supervisor having more than 3 people working under them.**

```
SELECT SupCode, COUNT(*)  
FROM Emp  
WHERE GradeCode < 10  
AND SupCode IN  
(SELECT SupCode  
FROM Emp  
GROUP BY SupCode  
HAVING COUNT(*) > 3 )  
GROUP BY Supcode;
```

SUPCOD	COUNT (*)
7698	1
7839	6

# EXISTS

47

List employees who did not get any promotion since 1990.

```
SELECT    EmpCode, EmpName, DeptCode  
FROM      Emp  
WHERE     NOT EXISTS  
          (promotion records for him since 1990);
```

# EXISTS

48

List employees who get any promotion since 1990.

```
SELECT EmpCode, EmpName, DeptCode  
FROM Emp E  
WHERE NOT EXISTS  
(SELECT *  
FROM History  
WHERE E.EmpCode = EmpCode  
AND changeDate >= date '1990-01-01');
```

# Nested Queries

49

List employees who were promoted to officer grade in 1995.

```
SELECT EmpCode, EmpName, DeptCode,  
       GradeCode, GradeLevel  
  FROM Emp  
 WHERE EmpCode IN ( SELECT EmpCode  
  FROM History  
 WHERE GradeCode >= 10  
 AND changeDate BETWEEN '01-Jan-95'  
                      AND '31-Dec-95');
```

EMPCOD	EMPNAME	DEPT	GRADECODE	GRADELEVEL
7369	Shah	PRCH	20	2

# Nested Queries

50

List employees who did not get any promotion since 1990.

```
SELECT EmpCode, EmpName, DeptCode  
FROM Emp  
WHERE NOT EXISTS ( SELECT *  
                   FROM History  
                   WHERE EMP.EmpCode = EmpCode  
                   AND changeDate > '01-Jan-90');
```

EMPCOD	EMPNAME	DEPT
7788	Khan	PRCH
7876	Patil	PRCH
7233	Kumaran	FACL

# Nested Queries

#

List the second highest Salary.

```
SELECT MAX (basic+allow-deduct) as takehome  
FROM   SALARY  
WHERE  (basic+allow-deduct)  
NOT IN (SELECT MAX (basic+allow-deduct)  
        FROM SALARY);
```

# Other DML commands

---

- INSERT
- UPDATE
- DELETE
- MERGE

# INSERT one complete row

**51**

Promote Shah as Salesman.

```
INSERT INTO History  
VALUES ('7369','01-Aug-96','SLMN','12',5000);
```

# INSERT one partial row

**52**

**Employ Hussein as a temporary employee.**

```
INSERT INTO Emp  
(EmpCode, EmpName, Basic)  
VALUES  
('9123', 'Hussein', 250);
```

# INSERT thru' Subquery

53

Update the SALARY table for the month.

```
INSERT INTO Salary  
SELECT EmpCode, CURRENT_DATE,  
      Basic, Basic*1.5, Basic*0.3  
FROM   Emp;
```

# Delete one row

**54**

**Delete employee record of Kaul.**

```
DELETE FROM Emp  
WHERE EmpCode = '7934';
```

# Bulk Delete

**55**

**Delete all employee records of Filing Department.**

```
DELETE FROM Emp  
WHERE DeptCode = 'FLNG';
```

# Delete entire Table contents

**56**

Delete the entire contents of **SALARY** table.

**DELETE FROM Salary;**

# Update one Row

**57**

Promote Gupta as Manager(Exports).

```
UPDATE Emp  
SET GradeCode = '4',  
DesigCode = 'MNGR',  
Basic = 15000  
WHERE EmpCode = '7654';
```

# Bulk Update

58

Raise the budget by 25% for all the departments except Facilities department.

```
UPDATE      Dept
SET        DeptBudget = DeptBudget * 1.25
WHERE      DeptCode != 'FACL';
```

# DDL commands

---

- CREATE
- ALTER
- DROP
- TRUNCATE

# CREATE TABLE

59

Create Department table

```
CREATE TABLE Dept (
    DeptCode      varchar2 (4)
                  constraint dept_pk primary key,
    DeptName      varchar2 (25) NOT NULL,
    DeptManager   varchar2 (6),
    DeptBudget    number NOT NULL);
```

# CREATE TABLE

60

Create Employee table

```
CREATE TABLE Emp (
    EmpCode      varchar2(6)
                  constraint emp_pk primary key,
    EmpName      varchar2(20) not null,
    DeptCode     varchar2 (4)
                  constraint emp_dept_rc
                  references dept(deptcode),
```

*Cntd ....*

# CREATE TABLE

BirthDate	Date not null,
JoinDate	Date not null,
Sex	char(1) not null check (sex in ('M', 'F')),
DesigCode	char(4) not null constraint emp_desig_rc references desig(DesigCode) ,
SupCode	char(6) constraint emp_sup_rc references emp(EmpCode) ,
GradeCode	number(2),
Basic	number(5));

# CREATE TABLE

61

Create Grade table

```
CREATE TABLE Grade (
    GradeCode      number(2) not null,
    GradeLevel     number    not null,
    Basic          number(5) not null,
    constraint grade_pk
    primary key (GradeCode,GradeLevel)) ;
```

Table-level constraint

# CREATE TABLE - Copying table structure/data

61

## Create Grade table

**CREATE TABLE GradeNew Like Grade;**

This will copy the structure and indices, but not the data:

**CREATE TABLE GradeNew AS SELECT \* FROM Grade;**

This will copy the data and the structure, but not the indices

**INSERT INTO GradeNew SELECT \* FROM Grade;**

To Make structure of new table similar to previous

# Create Table – Auto increment

- The attribute to use when you want to assign a sequence of numbers automatically to a field
- A table in MySQL can only contain one AUTO\_INCREMENT column

```
CREATE TABLE contacts
( contact_id INT(11) NOT NULL AUTO_INCREMENT,
last_name VARCHAR(30) NOT NULL,
first_name VARCHAR(25),
birthday DATE,
CONSTRAINT contacts_pk PRIMARY KEY (contact_id) );
```

# Delete a table from the database :

**62**

**Drop the entire SALARY table.**

```
DROP TABLE Salary;
```

# ALTER TABLE

63

Drop column 'basic' from Grade

```
ALTER TABLE Grade  
DROP COLUMN Basic;
```

# ALTER TABLE

**64**

**Add column 'GradeInc' to Grade**

```
ALTER TABLE Grade  
ADD COLUMN GradeInc number;
```

# ALTER TABLE

65

Increase the size of the field SupCode in Emp table

```
ALTER TABLE Emp  
MODIFY COLUMN SupCode char(10);
```

# Updating Rows in a Table

- ☞ Specific row or rows are modified if you specify the WHERE clause.
- ☞ All rows in the table are modified if you omit the WHERE clause.
- ☞ Update emp set deptcode='SALE' where deptcode='6569'

# Database Constraints

- Default
  - The DEFAULT constraint is used to set a default value for a column.
  - The default value will be added to all new records, if no other value is specified.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    City varchar(255) DEFAULT 'Delhi'
);
```

# Database Constraints

- Unique
  - ```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

# Database Constraints

- Not Null
  - By default, a column can hold NULL values.
  - The NOT NULL constraint enforces a column to NOT accept NULL values.
  - This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

# Database Constraints

- Check
  - The CHECK constraint is used to limit the value range that can be placed in a column.
  - If you define a CHECK constraint on a column it will allow only certain values for this column.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CHECK (Age>=18)
);
```

# Deleting Rows Based on Another Table

```
DELETE FROM employees  
WHERE department_id =(SELECT department_id  
FROM departments WHERE  
department_name LIKE '%Public%');
```

👉 1 row deleted.

# Deleting Rows: Integrity Constraint Error

```
DELETE FROM  
departments
```

You cannot delete a row that contains a primary key that is used as a foreign key in another table

```
WHERE    department_id  
= 60;
```

```
DELETE FROM departments  
*
```

ERROR at line 1:

```
ORA-02292: integrity constraint (HR.EMP_DEPT_FK)  
violated - child record found
```

# TRUNCATE

**66**

**Remove all tuples from table Salary**

**TRUNCATE TABLE Salary;**

# Content

- Indexes
- Benefit of Indexes
- Type of Indexes
- Temporary Tables
- ACID Properties
- Concept of Database Instance and Schema
- MySQL Storage Engines (InnoDB, MyISAM and others)

# Index

- Technique to speed up the queries
- Can be created using one or more columns
- While creating index, it should be taken into consideration which all columns will be used to make SQL queries and create one or more indexes on those columns.
- Indexes are also a type of tables, which keep primary key or index field and a pointer to each record into the actual table.
- The users cannot see the indexes, they are just used to speed up queries and will be used by the Database Search Engine to locate records very fast.

# Index - Need

## Scenario :

A contact book that contains names and mobile numbers of the user.

To find the mobile number of “Kelvin”. If the contact book is an unordered format, sequential scan needs to be performed until the desired name is found. This type of searching name is known as sequential searching.

# Index - Need

```
mysql> SELECT      EmpName, DeptName, dept.DeptCode FROM emp,dept WHERE emp.Dept
Code = dept.DeptCode;
+-----+-----+-----+
| EmpName | DeptName | DeptCode |
+-----+-----+-----+
| Reddy   | Accounts | ACCT     |
+-----+-----+-----+
1 row in set (0.05 sec)
```

# Benefit of Indexes

- Finding a row for a particular employee and corresponding department requires the full table scan requires examination of each row in the table to see whether it matches the desired value.
- This involves a full table scan, which is slow, as well as tremendously inefficient if the table is large but contains only a few rows that match the search criteria

# Benefit of Indexes

- Indexes are used to speed up searches for rows matching terms of a WHERE clause
- Rows that match rows in other tables when performing joins.
- For queries that use the MIN() or MAX() functions, the smallest or largest value in an indexed column can be found quickly without examining every row.
- Indexes can be used to perform sorting and grouping operations quickly for ORDER BY and GROUP BY clauses.

# Choosing Indexes

- Index columns that you use for searching, sorting, or grouping, not columns you select for output
  - The columns that appear in WHERE clause, columns named in join clauses, or columns that appear in ORDER BY or GROUP BY clauses.
  - Columns that appear only in the output column list following the SELECT keyword are not good candidates:

# Choosing Indexes

```
SELECT
    col_a                                ← not a candidate
FROM
    tbl1 LEFT JOIN tbl2
    ON tbl1.col_b = tbl2.col_c ← candidates
WHERE
    col_d = expr;                         ← a candidate
```

# Choosing Indexes

- Consider column cardinality.
  - Indexes work best for columns that have a high cardinality relative to the number of rows in the table (that is, columns that have many unique values and few duplicates).
  - For a column that contains many different age values, an index readily differentiates rows.
  - For a column that is used to record gender and contains only the two values 'M' and 'F', an index will not help.
    - Under these circumstances, the index might never be used at all, because the query optimizer generally skips an index in favor of a full table scan if it determines that a value occurs in a large percentage of a table's rows.

# Choosing Indexes

- **Index short values.**
  - Use smaller data types when possible.
  - Don't use a BIGINT column if a MEDIUMINT is large enough to hold the values you need to store, and don't use CHAR(100) if none of your values are longer than 25 characters.
  - Shorter values can be compared more quickly, so index lookups are faster.
  - Smaller values result in smaller indexes that require less disk I/O.
  - With shorter key values, index blocks in the key cache hold more key values.

# Choosing Indexes

- **Take advantage of leftmost prefixes.**
- A composite index serves as several indexes because any leftmost set of columns in the index can be used to match rows.
- Such a set is called a “leftmost prefix.”
- Suppose that you have a table with a composite index on columns named state, city, and zip.
- Rows in the index are sorted in state/city/zip order, so they’re automatically sorted in state/city order and in state order as well.
- This means that MySQL can take advantage of the index even if you specify only state values in a query, or only state and city values.
- Thus, the index can be used to search the following combinations of columns: state, city, zip
- state, city
- state
- if you search by city or by zip, the index isn’t used.
- If you’re searching for a given state and a particular ZIP code, the index can’t be used for the combination of values

# Choosing Indexes

- **Don't over-index.**
- Every additional index takes extra disk space and hurts performance of write operations, as has already been mentioned.
- Indexes must be updated and possibly reorganized when you modify the contents of your tables, and the more indexes you have, the longer this takes.
- If you have an index that is rarely or never used, you'll slow down table modifications unnecessarily.
- In addition, MySQL considers indexes when generating an execution plan for retrievals.
- Creating extra indexes creates more work for the query optimizer.
- It's also possible (if unlikely) that MySQL will fail to choose the best index to use when you have too many indexes.
- If you're thinking about adding an index to a table that is already indexed, consider whether the index you're considering adding is a leftmost prefix of an existing multiple-column index.
  - For example, if you already have an index on state, city, and zip, there is no point in adding an index on state.

# Index - Issues

- The INSERT and UPDATE statements take more time on tables having indexes, whereas the SELECT statements become fast on those tables.
- The reason is that while doing insert or update, a database needs to insert or update the index values as well.

# Index- Type

- **Unique index**
  - This disallows duplicate values.
  - For a single-column index, this ensures that the column contains no duplicate values.
  - For a multiple-column (composite) index, it ensures that no combination of values in the columns is duplicated among the rows of the table.
- **Non-unique index**
  - This gives you indexing benefits but allows duplicates.
- **FULLTEXT index**
  - Used for performing full-text searches.
  - This index type is supported only for MyISAM tables.

# Index- Type

- **SPATIAL index :**
  - These can be used only with MyISAM tables for the spatial data types(storing the coordinates)
- **HASH index**
  - This is the default index type for MEMORY tables, although you can override the default to create BTREE indexes instead.

# Index command

- CREATE INDEX index\_name ON tbl\_name (index\_columns);
- CREATE UNIQUE INDEX index\_name ON tbl\_name (index\_columns);
- To drop an index, use either a DROP INDEX or an ALTER TABLE statement.
  - DROP INDEX index\_name ON tbl\_name;
  - ALTER TABLE tbl\_name DROP INDEX index\_name;
- Show index
  - show index from emp \G;
- INDEX, you must name the index to be dropped:
- DROP INDEX index\_name ON tbl\_name;
- multiple indexes can not be created with a single statement.

# Index command

- ```
CREATE TABLE tbl_name
(
    ... column definitions ...
    INDEX index_name (index_columns),
    UNIQUE index_name (index_columns),
    PRIMARY KEY (index_columns),
    FULLTEXT index_name (index_columns),
    SPATIAL index_name (index_columns),
    ...
);
```
- If a column is dropped that is a part of an index, MySQL removes the column from the index as well.
- If you drop all columns that make up an index, MySQL drops the entire index

# Temporary Tables

- If you add the TEMPORARY keyword to a table-creation statement, the server creates a temporary table that disappears automatically when your connection to the server terminates
- DROP TABLE statement is not needed to get rid of the table
- A TEMPORARY table is visible only to the client that creates the table.
  - Different clients can each create a TEMPORARY table with the same name and without conflict because each client sees only the table that it created.

# Temporary Tables

- The name of a TEMPORARY table can be the same as that of an existing permanent table.
  - This is not an error, nor does the existing permanent table get clobbered.
  - Instead, the permanent table becomes hidden (inaccessible) to the client that creates the TEMPORARY table while the TEMPORARY table exists.
  - Suppose that you create a TEMPORARY table named member in the sampdb database.
  - The original member table becomes hidden, and references to member refer to the TEMPORARY table.
  - If you issue a DROP TABLE member statement, the TEMPORARY table is removed and the original member table “reappears.”
  - If you disconnect from the server without dropping the TEMPORARY table, the server automatically drops it for you.

# Temporary Tables

- CREATE TEMPORARY TABLE tbl\_name;
- DROP TEMPORARY TABLE tbl\_name

# MySQL Storage Engines

- MySQL supports multiple storage engines/table handlers
- Each storage engine implements tables that have a specific set of properties or characteristics.

# MySQL Storage Engines

---

Storage Engine	Description
ARCHIVE	Archival storage (no modification of rows after insertion)
BLACKHOLE	Engine that discards writes and returns empty reads
CSV	Storage in comma-separated values format
EXAMPLE	Example (“stub”) storage engine
Falcon	Transactional engine
FEDERATED	Engine for accessing remote tables
InnoDB	Transactional engine with foreign keys
MEMORY	In-memory tables
MERGE	Manages collections of MyISAM tables
MyISAM	The default storage engine
NDB	The engine for MySQL Cluster

---

# MySQL Storage Engines

```
mysql> SHOW ENGINES\G
***** 1. row *****
Engine: ARCHIVE
Support: YES
Comment: Archive storage engine
Transactions: NO
XA: NO
Savepoints: NO
***** 2. row *****
Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
XA: NO
Savepoints: NO
***** 3. row *****
Engine: MRG_MYISAM
Support: YES
Comment: Collection of identical MyISAM tables
Transactions: NO
XA: NO
Savepoints: NO
***** 4. row *****
Engine: FEDERATED
Support: NO
Comment: Federated MySQL storage engine
Transactions: NULL
XA: NULL
Savepoints: NULL
```

# MySQL Storage Engines

- The value in the Support column is YES or NO to indicate that the engine is or is not available,
- DISABLED if the engine is present but turned off,
- DEFAULT for the storage engine that the server uses by default.
- The engine designated as DEFAULT should be considered available.
- The Transactions column indicates whether an engine supports transactions.
- XA and Savepoints indicate whether an engine supports distributed transactions and partial transaction rollback.

# InnoDB Storage Engine

- Transaction-safe tables with commit and rollback.
- Savepoints can be created to enable partial rollback.
- Automatic recovery after a crash.
- Foreign key and referential integrity support, including cascaded delete and update.
- Row-level locking and multi-versioning for good concurrency performance under query mix conditions that include both retrievals and updates.

# MyISAM Storage Engine

- MyISAM provides key compression.
- Uses compression when storing runs of successive similar string index values.
- MyISAM provides more features for AUTO\_INCREMENT columns than do other storage engines.
- MyISAM tables also have a flag indicating whether a table was closed properly when last used.
- If the server shuts down abnormally or the machine crashes, the flags can be used to detect tables that need to be checked.

# Database Technologies

Session3

# Contents

- Data Redundancy
- Data Anomalies
- Database Constraints (Unique, Not Null, Foreign Key, Default, Check\*)
- Functional Dependency
- Normalization
- Need for Normalization
- Normal Forms (1st NF, 2nd NF, 3rd NF, BCNF) with examples, Introduction to 4th and 5th NF
- DML (INSERT/UPDATE/DELETE)

# Normalization

- Normalization is a data analysis technique to design a database system.
  - An analytical technique used during logical database design
  - Offers a strategy for constructing relations and identifying keys
- Normalization is a technique for producing relational schema with the following properties:
  - No Information Redundancy
  - No Update Anomalies

# Why Normalize - 1

emp code	emp name	join date	dept code	dept name	dept mngr	dept bdgt
7369	shah	17-Dec-80	prch	purchase	roy	5
7499	ray	20-Feb-81	prch	purchase	roy	5
7521	jain	02-Apr-82	prch	purchase	roy	5
7654	gupta	28-Sep-79	info	infoserv	rao	6.5

redundancy

# Why Normalize - 2

emp code	emp name	join date	dept code	dept name	dept mngr	dept bdgt
7369	shah	17-Dec-80	prch	purchase	roy	5
7499	ray	20-Feb-81	prch	purchase	roy	5
7521	jain	02-Apr-82	prch	purchase	roy	5
7654	gupta	28-Sep-79	info	infoserv	rao	6.5

attributes are lost because of the deletion  
of other attributes

deletion  
anomaly

# Why Normalize - 2

emp code	emp name	join date	dept code	dept name	dept mngr	dept bdgt
7369	shah	17-Dec-80	prch	purchase	roy	5
7499	ray	20-Feb-81	prch	purchase	roy	5
7521	jain	02-Apr-82	prch	purchase	roy	5



deletion  
anomaly

# Why Normalize - 3

emp code	emp name	join date	dept code	dept name	dept mngr	dept bdgt
7369	shah	17-Dec-80	prch	purchase	roy	5
7499	ray	20-Feb-81	prch	purchase	roy	5
7521	jain	02-Apr-82	prch	purchase	roy	5
7654	gupta	28-Sep-79	info	infoserv	rao	6.5

Update Anomaly exists when one or more instances of duplicated data is updated, but not all.

update  
anomaly

# Why Normalize - 3

emp code	emp name	join date	dept code	dept name	dept mngr	dept bdgt
7369	shah	17-Dec-80	prch	purchase	apte	5
7499	ray	20-Feb-81	prch	purchase	apte	5
7521	jain	02-Apr-82	prch	purchase	apte	5
7654	gupta	28-Sep-79	info	infoserv	rao	6.5

update  
anomaly

# Why Normalize - 4

emp code	emp name	join date	dept code	dept name	dept mngr	dept bdgt
7369	shah	17-Dec-80	prch	purchase	roy	5
7499	ray	20-Feb-81	prch	purchase	roy	5
7521	jain	02-Apr-82	prch	purchase	roy	5
			info	infoserv	rao	6.5

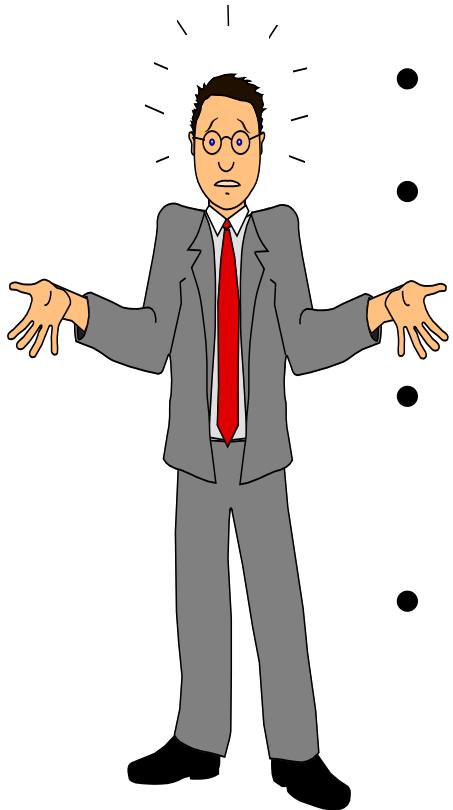
Insert Anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attributes

insertion  
anomaly

# Normalisation Stages

- Process involves applying a series of tests on a relation to determine whether it satisfies or violates the requirements of a given normal form.
  - When a test fails, the relation is decomposed into simpler relations that individually meet the normalization tests.
  - The higher the normal form the less vulnerable to update anomalies

# Normal Form??



- “restriction” on a relation
- a relation that satisfies certain rules/conditions
- a relation that exhibits certain properties
- depending on the conditions it satisfies/the properties it exhibits, the relation is said to be in the “n<sup>th</sup> Normal Form”

# Normal Forms

- 1 NF
- 2 NF
- 3 NF
- BCNF
- 4NF
- 5NF

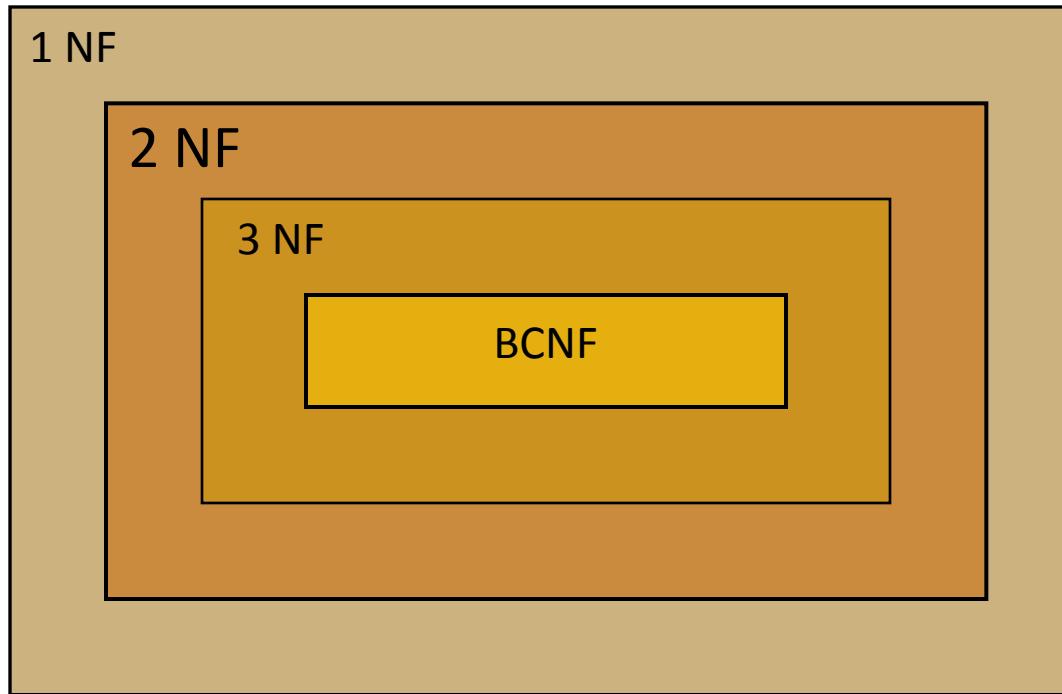


*Functional Dependency*

*Multi-valued Dependency*

*Join Dependency*

# Hierarchy of Normal Forms



---

Normal Forms are **INCREMENTAL**

# Hierarchy of Normal Forms

- It must be emphasised here, that the definition of a Normal Form is INCREMENTAL.
- You cannot have some relation that is in X-normal form, but not in (X-1) Normal Form.
- If a relation is in 3 NF, it also has to be in 2NF, which means it is also in 1NF.

# Integrity Constraints

- Databases are structured stores of data
- Data must be accurate
  - Semantic accuracy v/s Syntactic accuracy
    - consider an attribute **age** of type **int**. Any integer would be **syntactically correct**; but if the attribute pertains to the age of say, drivers, then any value less than 18 would be **semantically incorrect**.
- Integrity constraints are the business rules of the problem-domain.

# Domain Constraints

- Integrity constraints guard against accidental damage to the database, by ensuring that authorized changes to the database do not result in a loss of data consistency.
- Domain constraints are the most elementary form of integrity constraint.

# Referential Integrity

- Ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.
- Formal Definition
  - Let  $r_1 (R_1)$  and  $r_2 (R_2)$  be relations with primary keys  $K_1$  and  $K_2$  respectively.
  - The subset  $\alpha$  of  $R_2$  is a foreign key referencing  $K_1$  in relation  $r_1$ , if for every  $t_2$  in  $r_2$  there must be a tuple  $t_1$  in  $r_1$ 
    - such that  $t_1 [K_1] = t_2 [\alpha]$ .
    - Referential integrity constraint:  $\Pi \alpha (r_2) \subseteq \Pi K_1 (r_1)$

# more on Integrity Constraints

- 3 types of Integrity Constraints are of interest:
  - Functional Dependencies (FD)
  - Multi-Valued Dependency (MVD)
  - Join-Dependency (JD)
- FDs are the most commonly used integrity constraints in normalization.

# Functional Dependencies

- Defines a constraint between two (sets of) attributes of a relation
- Require that the value for a certain set of attributes determines uniquely the value for another set of attributes.
- Written as “ $X \rightarrow Y$ ”\* (read “X determines Y”)

\* here, “X” is called the “determinant” and “Y”, the “consequent”

# FD: Example

- Consider a “Student” relation with the schema  $\{\text{S\_ID}, \text{Name}, \text{DateOfBirth}, \text{Address}, \text{City}\}$
- Also consider the “ExamResults” relation with the schema  $\{\text{S\_ID}, \text{ExamID}, \text{MaxMarks}, \text{ObtMarks}\}$
- Typical FDs that hold over these schemas:
  - $\{\text{S\_ID}\} \rightarrow \{\text{Name}, \text{DateOfBirth}, \text{Address}, \text{City}\}$
  - $\{\text{S\_ID}, \text{ExamID}\} \rightarrow \{\text{ObtMarks}\}$
  - $\{\text{ExamID}\} \rightarrow \{\text{MaxMarks}\}$

# FD: Defined

- Let  $R$  be a relation schema
    - $\alpha \subseteq R, \beta \subseteq R$
  - The functional dependency
    - $\alpha \rightarrow \beta$   
holds on  $R$  if and only if for any legal relations  $r(R)$ , whenever any two tuples  $t_1$  and  $t_2$  of  $r$  agree on the attributes  $\alpha$ , they also agree on the attributes  $\beta$ .
- That is,
- $$t_1[\alpha] = t_2[\alpha] \Rightarrow t_1[\beta] = t_2[\beta]$$
- $K$  is a superkey for relation schema  $R$  if and only if  $K \rightarrow R$
  - $K$  is a candidate key for  $R$  if and only if
    - $K \rightarrow R$ , and
    - for no  $\alpha \subset K, \alpha \rightarrow R$

# Trivial and nontrivial dependency

- Trivial
  - If an FD  $X \rightarrow Y$  holds where  $Y$  subset of  $X$ , then it is called a trivial FD.
  - If the right hand side (dependent) is subset of left hand side (determinant)
- Non-trivial
  - If an FD  $X \rightarrow Y$  holds where  $Y$  is not subset of  $X$ , then it is called non-trivial FD.

# Axioms of Functional Dependency - I

- Given a set F set of functional dependencies, there are certain there functional dependencies that are logically implied by F .
- The set of all functional dependencies logically implied by F is the closure of F ( $F^+$ )
- Axiom of Reflexivity**  
If  $Y \subseteq X$ , then  $X \rightarrow Y$
- Axiom of Augmentation**  
If  $X \rightarrow Y$  and  $W \rightarrow Z$ , then  $XW \rightarrow YZ$
- Axiom of Transitivity**  
If  $X \rightarrow Y$  and  $Y \rightarrow Z$ , then  $X \rightarrow Z$

# Axioms of Functional Dependency - II

- **Union Rule**

If  $X \rightarrow Y$  and  $X \rightarrow Z$ , then  $X \rightarrow YZ$

- **Decomposition Rule**

If  $X \rightarrow YZ$ , then  $X \rightarrow Y$  and  $X \rightarrow Z$

# Attribute Set Closure

- Closure of an attribute set  $\alpha$ , denoted by  $\alpha^+$ , are:
  - the set of attributes that are determined by  $\alpha$
- OR
- the set of attributes that are functionally dependent on  $\alpha$
- Attribute set closure helps in identifying the super keys in a relation

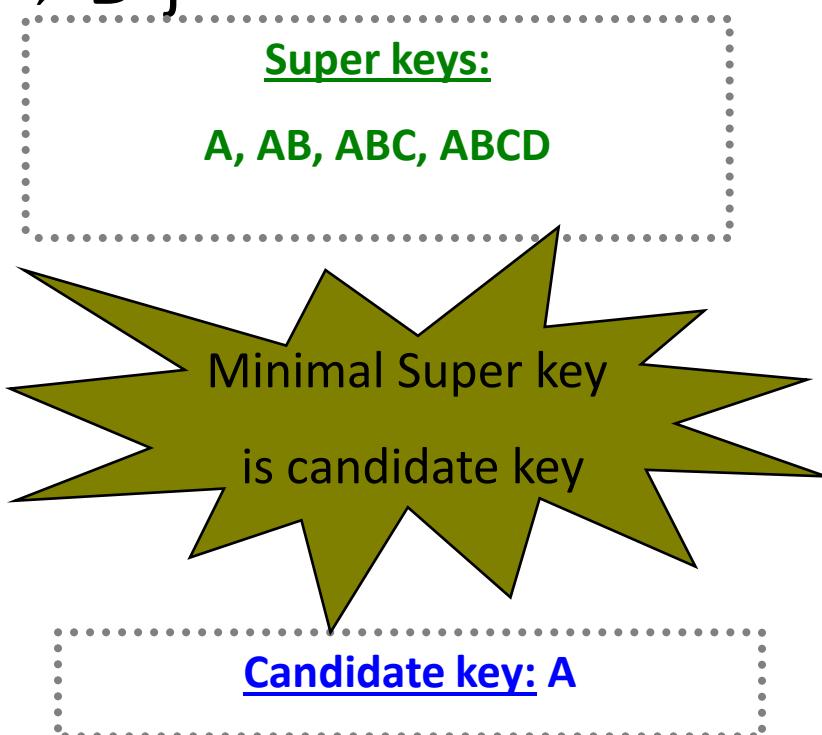
# Attribute Set Closure - Example

Given: Relation R(A, B, C, D)

FDs {  $A \rightarrow B$ ,  $B \rightarrow C$ ,  $C \rightarrow D$  }

Here,

- $A^+ = \{A, B, C, D\}$
- $B^+ = \{B, C, D\}$
- $(AB)^+ = \{A, B, C, D\}$
- $(CD)^+ = \{C, D\}$
- $D^+ = \{D\}$



# Decomposition

- Most common method of normalization; involves the splitting of original schema into subset schemas.
- Decomposition is not arbitrary; subset schemas (and remainder if any, of original schema) must conform to certain rules
- Decide whether a particular relation R is in “good” form.
- In the case that a relation R is not in “good” form, decompose it into a set of relations of  $R_1, R_2, \dots, R_n$  such that
  - each relation is in good form
  - the decomposition is a lossless-join decomposition

# Decomposition : *Desirable Properties*

- Lossless-Join
  - the join of the subset schemas when computed, must yield exactly the original relation - not a tuple more, not a tuple less.
- Dependency Preservation
  - decomposition should be such that none of the functional dependencies are lost

for more info:

An Introduction to Database Systems, C J Date, Ch. 10, 11

# Lossless-Join Decomposition



Let  $R\{A, B, C\}$  be a relation.

If  $R$  is equal to the join of the projections  $R1\{A, B\}$  and  $R2\{A, C\}$ , then this decomposition is Lossless.

# Lossless-Join Decomposition



Let  $\mathbf{R}$  be a relation

Let  $\mathbf{R}$  be decomposed into two relations  
 $\mathbf{R1}$  and  $\mathbf{R2}$ .

If  $[\mathbf{R1} \cap \mathbf{R2}]^+ = \{\text{all attributes of } \mathbf{R1}\}$  or  
 $\{\text{all attributes of } \mathbf{R2}\}$ ,  
then the decomposition is Lossless.

# Lossless-Join Decomposition - Example

Consider the relation  $R(A, B, C, D)$  and FDs

$\{A \rightarrow B, B \rightarrow C, C \rightarrow D\}$ . Consider the decomposition of  $R$  into  $R1(A, B, C)$  and  $R2(B, C, D)$ . Is this decomposition lossless in nature?

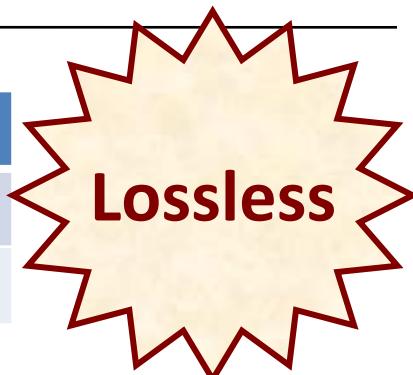
- Here  $R1 \cap R2 = \{B, C\}$
  - $(BC)^+ = \{B, C, D\} = R2$
  - Hence this decomposition is lossless.
-

# Example

S	STATUS	CITY
S3	30	Paris
S5	30	Athens

S	Status
S3	30
S5	30

S	CITY
S3	Paris
s5	Athens



Lossless

S	STATUS
S3	30
s5	30

STATUS	CITY
30	Paris
30	Athens



Lossy

Which supplier has which city

# Lossless-Join Decomposition

Consider the relation

**INFO { ID ,City ,Ppln }**

with the FDs

$ID \rightarrow City$

$City \rightarrow Ppln$

Possible decompositions:

+ { {ID,City}, {ID,Ppln} }

+ { {ID,City}, {City,Ppln} }

+ { {ID,Ppln}, {City,Ppln} }

ID	City	Ppln
41	Bangalore	5,00,000
42	Trichy	2,50,000
43	Coimbatore	5,00,000
44	Mumbai	8,00,000
45	Mumbai	8,00,000
46	Bangalore	5,00,000
47	Bangalore	5,00,000

# Decomposing “INFO”

Decomposed into { {ID,Ppln}, {City,Ppln} }

INFO\_Ppn

ID	Ppln
41	5,00,000
42	2,50,000
43	5,00,000
44	8,00,000
45	8,00,000
46	5,00,000
47	5,00,000

City\_Ppn

City	Ppln
Bangalore	5,00,000
Trichy	2,50,000
Coimbatore	5,00,000
Mumbai	8,00,000

# “Recomposing INFO”

$$\text{INFO} = \text{INFO\_Ppn} \bowtie \text{City\_Ppn}$$

INFO\_Ppn

ID	City	Ppln
41	Bangalore	5,00,000
41	Coimbatore	5,00,000
42	Trichy	2,50,000
43	Coimbatore	5,00,000
43	Bangalore	5,00,000
44	Mumbai	8,00,000
45	Mumbai	8,00,000

ID	City	Ppln
46	Bangalore	5,00,000
46	Coimbatore	5,00,000
47	Bangalore	5,00,000
47	Coimbatore	5,00,000

$\{\{\text{ID}, \text{Ppln}\},$   
 $\{\text{City}, \text{Ppln}\}\}$  was a lossy  
decomposition

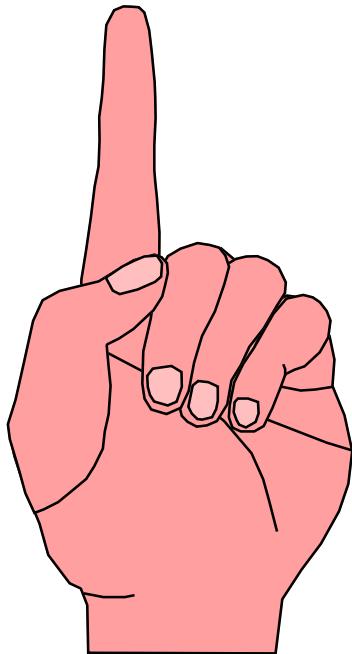
# Unnormalized

emp_data	emp code	emp name	grade code	grade stage	desig code	change date	grade descr	desig descr
	7369	shah	12	3	slasst	20-Feb-82	Asst_A	Sls Asst
			---	---	---	---	---	---
			---	---	---	---	---	---
	7499	ray	23	2	supr	17-Jan-95	Offcr A	Superint.
			---	---	---	---	---	---
			---	---	---	---	---	---

multi-valued attributes

not  
in  
1NF!

# Unnormalized 1NF



- Eliminate variable repeating fields and groups so that all attributes take atomic values

---

A relation is said to be in “first normal form” (1NF) if and only if all its attributes assume only atomic(indivisible) values.

---

# Unnormalized → 1NF

emp\_data

emp code	emp name	grade code	grade stage	desig code	change date	grade descr	desig descr
7369	shah	12	3	slasst	20-Feb-82	Asst_A	Sls Asst
		---	---	---	---	---	---
		---	---	---	---	---	---
7499	ray	23	2	supr	17-Jan-95	Offcr A	Superint.
		---	---	---	---	---	---
		---	---	---	---	---	---

# Unnormalized 1NF

emp

emp code	emp name
7369	shah
7499	

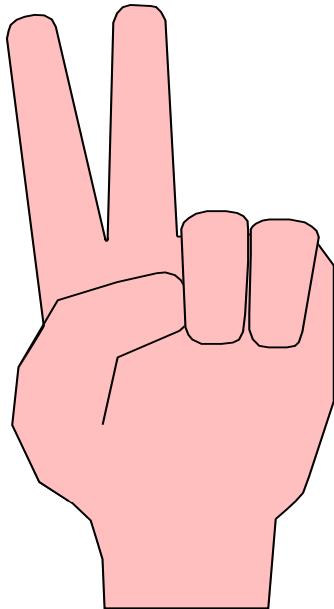
in  
1NF!

emp\_hist

emp code	grade code	grade stage	desig code	change date	grade descr	desig descr
7369	12	3	slas	20-Feb-82	Asst_A	Sls_Asst
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7499	23	2	supr	17-Jan-95	Offcr A	Superint.
7499	---	---	---	---	---	---
7499	---	---	---	---	---	---
7499	---	---	---	---	---	---

in  
1NF!

# 1NF → 2NF



Eliminate fields that are facts about only a *subset* of the key so that all non-key domains are fully functionally dependent on the primary key.

---

A relation is said to be in 2NF if and only if it is in 1NF and every non-key attribute is **fully functionally dependent (No partial dependency)** on any key.

---

# 1NF

emp

emp\_hist

Grade\_descr

*is not fully functionally dependent* on the primary key.

not in  
2NF!

emp\_hist

emp code	grade code	grade stage	desig code	change date	grade descr	desig descr
7369	12	3	slas	20-Feb-82	Asst_A	SIs_Asst
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7499	23	2	supr	17-Jan-95	Offcr A	Superint.
7499	---	---	---	---	---	---
7499	---	---	---	---	---	---
7499	---	---	---	---	---	---

# Eliminating partial dependencies

emp\_hist

1NF

emp

emp\_hist

emp code	grade code	grade stage	desig code	change date	grade descr	desig descr
7369	12	3	slas	20-Feb-82	Asst_A	SIs_Asst
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7369	---	---	---	---	---	---
7499	23	2	supr	17-Jan-95	Offcr A	Superint.
7499	---	---	---	---	---	---
7499	---	---	---	---	---	---
7499	---	---	---	---	---	---

# Eliminating partial dependencies

emp\_hist

1NF

emp

emp\_hist

emp code	grade code	grade stage	desig code	change date	desig descr
7369	12	3	slas	20-Feb-82	SIs_Asst
7369	---	---	---	---	---
7369	---	---	---	---	---
7369	---	---	---	---	---
7369	---	---	---	---	---
7499	23	2	supr	17-Jan-95	Superint.
7499	---	---	---	---	---
7499	---	---	---	---	---
7499	---	---	---	---	---

# 1NF

emp

emp\_hist

## grade

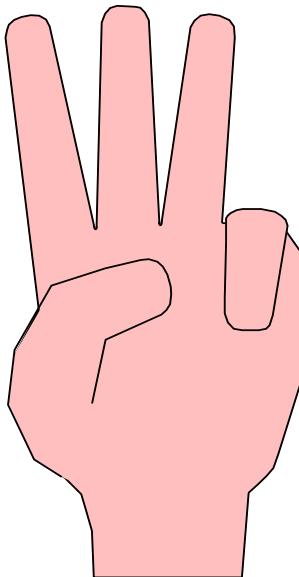
grade code	grade descr
12	Asst_A
23	Offcr A



## emp\_hist

emp code	grade code	grade stage	desig code	change date	desig descr
7369	12	3	slas	20-Feb-82	SIs_Asst
7369	---	---	---	---	---
7369	---	---	---	---	---
7369	---	---	---	---	---
7369	---	---	---	---	---
7499	23	2	supr	17-Jan-95	Superint.
7499	---	---	---	---	---
7499	---	---	---	---	---
7499	---	---	---	---	---

2NF → 3NF



Eliminate non-key fields  
that are facts about other  
non-key fields, so that all  
non-key domains are  
mutually independent.

---

A relation is said to be in 3NF if and only if it is in 2NF and for every  
nonkey attribute is non transitively dependent on primary key.

---

## 2NF

- emp
- emp\_hist
- grade

*desig\_descr* is  
*transitively dependent* on the  
primary key.



emp\_hist

emp code	grade code	grade stage	desig code	change date	desig descr
7369	12	3	slas	20-Feb-82	Sl Asst
7369	---	---	---	---	---
7369	---	---	---	---	---
7369	---	---	---	---	---
7369	---	---	---	---	---
7499	23	2	supr	17-Jan-95	Superint.
7499	---	---	---	---	---
7499	---	---	---	---	---
7499	---	---	---	---	---

# Eliminating Transitive Dependencies

2NF

- emp
- emp\_hist
- grade

emp\_hist

emp code	grade code	grade stage	desig code	change date	desig descr
7369	12	3	slas	20-Feb-82	SIs_Asst
7369	---	---	---	---	---
7369	---	---	---	---	---
7369	---	---	---	---	---
7369	---	---	---	---	---
7499	23	2	supr	17-Jan-95	Superint.
7499	---	---	---	---	---
7499	---	---	---	---	---
7499	---	---	---	---	---

# Eliminating Transitive Dependencies

2NF

- emp
- emp\_hist

- grade

desig

desig code	desig descr
slas	Sls_Asst
supr	Superint.

emp\_hist

emp code	grade code	grade stage	desig code	change date
7369	12	3	slas	20-Feb-82
7369	---	---	---	---
7369	---	---	---	---
7369	---	---	---	---
7369	---	---	---	---
7499	23	2	supr	17-Jan-95
7499	---	---	---	---
7499	---	---	---	---
7499	---	---	---	---

in  
3NF

# 3NF BCNF\*

- A relation schema R is in BCNF with respect to a set F of functional dependencies if for all functional dependencies in F+ of the form  $\alpha \rightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$  at least one of the following holds:
  - $\alpha \rightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$ )
  - $\alpha$  is a superkey for R

\*BCNF = Boyce Codd Normal Form

3NF → BCNF\*



Eliminate key fields that are facts about other (key) fields so that every determinant becomes a superkey.

\*BCNF = Boyce Codd Normal Form

---

A relation is said to be in BCNF if and only if, for every nontrivial FD  $A \rightarrow B$ , 'A' is a superkey.

---

# Problem-Domain (BCNF)

*Consider a sales-management scenario:*

- There are several product categories.
- Products are sold in several cities.
- Each city has several agents.
- Each product category is sold in each city by several retail outlets.
- A given product category is distributed in a given city by one and only one agent.
- A given agent will operate in one and only one city.
- A given agent can stock & sell more than one product category for the same city.

# Problem-domain (BCNF)

An appropriate schema would be

Supply {Product, City, Outlets, Agent}

Supply

Product	City	Outlets	Agent
Noodles	Margao	155	Daulat
Chocolates	Margao	110	Keshav
Baby Foods	Margao	235	Daulat
Ketchup	Panjim	163	Magsons
Noodles	Panjim	195	Aletta
Chocolates	Vasco	102	Pranav

FDs:

$$\{P, C\} \rightarrow \{O, A\}$$

$$\{A\} \rightarrow \{C\}$$

$$\boxed{\{A, P\} \rightarrow \{C\}}$$



3NF → BCNF

Supply

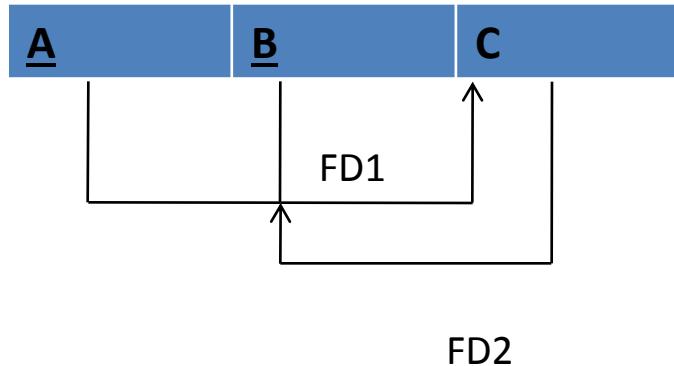
Product	City	Outlets
Noodles	Margao	155
Chocolates	Margao	110
Baby Foods	Margao	235
Ketchup	Panjim	163
Noodles	Panjim	195
Chocolates	Vasco	102

Agent	City
Daulat	Margao
Keshav	Margao
Magsons	Panjim
Aletta	Panjim
Pranav	Vasco

A\_Product

Agent	Product
Daulat	Noodles
Daulat	Baby Foods
Keshav	Chocolates
Magsons	Ketchup
Aletta	Noodles
Pranav	Chocolates





- Schematic relation in 3NF but not in BCNF  
B being prime attribute (member of some candidate key)

# Beyond BCNF...

- CTX is in BCNF, it has no functional dependency since it is an all key relation
- a course is taught by multiple teachers
- the course uses multiple textbooks
- there is redundancy

## Schema CTX

CTX

course	teachers	texts
Phy	Green	T1
Phy	Green	T2
Phy	Brown	T1
Phy	Brown	T2
Math	Green	T1
Math	Green	T2

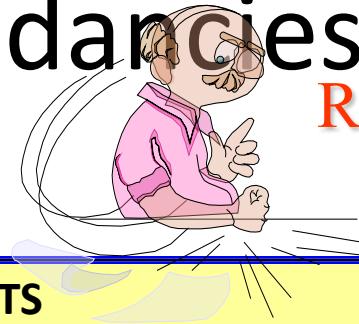
# More redundancies ...

COURSE	TEACHERS	TEXTS
Physics	{ Prof. Green Prof. Brown }	{ Basic Mechanics Principles of Optics }
Math	{ Prof. White }	{ Basic Mechanics Vector Analysis Trigonometry }

Note: There are no FDs in this relation.

# More redundancies ...

Redundancy !!



CTX

COURSE	TEACHERS	TEXTS
Physics	Prof. Green	Basic Mechanics
	Prof. Green	Principles of Optics
	Prof. Brown	Basic Mechanics
	Prof. Brown	Principles of Optics
Math	Prof. White	Basic Mechanics
	Prof. White	Vector Analysis
	Prof. White	Trigonometry

Constraint: IF tuples  $(c, t1, x1), (c, t2, x2)$  both appear

**THEN tuples  $(c, t1, x2), (c, t2, x1)$  both appear also**

- Redundancy :
  - TO add information that a physics course can be taught by a new teacher
    - Insert two new tuples

# Multi-valued Dependency

IF tuples  $(c, t1, x1), (c, t2, x2)$  both appear

THEN tuples  $(c, t1, x2), (c, t2, x1)$  both appear also

**Course**      → **teachers**

**Course**      →→ **texts**

They are independent MVDs

# Multi-valued Dependency

- Defines a constraint between two (sets of) attributes of a relation
  - Constraint in turn, defined by the semantics of the problem-domain
  - Helps identify redundancy that cannot be identified by mere FD analysis
  - MVDs are generalization of FDs
    - Every FD is MVD but converse is not true
- Written as “ $X \rightarrow Y$ ” (read “X multidetermines Y”)

- For a relation R (A,B,C) the MVD
  - $A \rightarrow\!\!\! \rightarrow B$  holds if and only if MVD  $A \rightarrow\!\!\! \rightarrow C$  also holds

# MVD: Defined

given a relation schema R, and two attribute sets X, Y such that

$$X \subseteq R, Y \subseteq R,$$

then, the MVD “ $X \rightarrow Y$ ” means

$\forall t_1, t_2, t_3, t_4 \in r$  ( ‘r’ is an instance of ‘R’)

the following is true

$$t_1[X] = t_2[X] = t_3[X] = t_4[X]$$

$$t_1[Y] = t_3[Y]$$

$$t_3[R-Y] = t_2[R-Y]$$

$$t_4[Y] = t_2[Y]$$

$$t_4[R-Y] = t_1[R-Y]$$

# BCNF 4NF

- A relation schema  $R$  is in 4NF with respect to a set  $D$  of functional and multivalued dependencies if for all multivalued dependencies in  $D^+$  of the form  $\alpha \twoheadrightarrow \beta$ , where  $\alpha \subseteq R$  and  $\beta \subseteq R$ , at least one of the following hold:
  - $\alpha \twoheadrightarrow \beta$  is trivial (i.e.,  $\beta \subseteq \alpha$  or  $\alpha \cup \beta = R$ )
  - $\alpha$  is a superkey for schema  $R$
- If a relation is in 4NF it is in BCNF

BCNF → 4NF

~~CTX~~

	course	teacher	text
Phy	Green	T1	
Phy	Green	T2	
Phy	Brown	T2	
Phy	Brown	T1	
Math	white	T3	
Math	White	T4	

CT

course	teacher
Phy	Green
Phy	Brown
Math	White

CX

course	text
Phy	T1
Phy	T2
Math	T3
Math	T4

In 4NF!

Eliminating independent, multi-valued facts

- Not in 4NF because the following non trivial multivalued dependency holds on them and Course is not a superkey

**Course** → **texts**

**Course** → **teachers**

- After decomposition the two relations are in 4NF because the MVD are trivial MVD(i.e their union produces the original relation)

# Normalization : a recap

- leads to a design that caters to ad-hoc queries.
- prevents update anomalies and data inconsistencies.
- facilitates data independence.
- penalises data retrieval.
- rule-based, but intricately influenced by semantics of the attributes.

# Normal Forms : a recap

1NF

**Eliminate repeating groups;** attributes must have only atomic values.

2NF

**Eliminate partial dependencies;** all non-key attributes must be functionally dependent on *entire* primary key, not on a subset thereof.

3NF

**Eliminate transitive dependencies;** all non-key domains must be mutually independent.

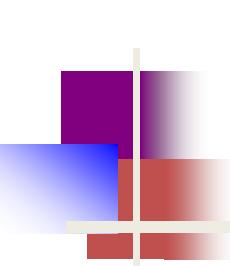
# Normal Forms : a recap

BCNF

**Eliminate FDs with prime attributes in the consequent;** all determinants must be super keys.

4NF

**Eliminate independent, multi-valued attributes.**



# References - I



## An Introduction to DB Systems (7/e)

C. J. Date

*Addison-Wesley (Pearson-Asia Education)*



## Database System Concepts (3/e)

Silberschatz, Korth, Sudarshan

*McGraw-Hill International Edition*



## Database Management Systems

Raghu Ramakrishnan

*McGraw-Hill International Edition*



## Fundamental of Database Systems

Elmasri

*Navathe*

# Views

- Virtual table based on the result-set of SQL statement and that is Stored in the database with some name.
- A view can contain all rows of a table or select rows from a table.
- A view can be created from one or many tables which depends on the written SQL query to create a view.
- Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more.
- Summarize data from various tables which can be used to generate reports.
- If table structure is changed then the view definition also has to be changed.

# Views

- **Simple View**

- A view based on only a single table, which doesn't contain GROUP BY clause and any functions.

- **Complex View**

- A view based on multiple tables, which contain GROUP BY clause and functions.

- **Inline View**

- A view based on a subquery in FROM Clause, that subquery creates a temporary table and simplifies the complex query.

- **Materialized View**

- A view that stores the definition as well as data. It creates replicas of data by storing it physically.

Features	Simple View	Complex View
No Of Tables	One	One or More
Contain function	NO	Yes
Contain group of data	No	Yes

# Views - Creation

**67**

Create a view for Employee-Age

```
create view empage(empcode,age)
as (select empcode, timestampdiff(year,birthdate,curdate())
from emp);
```

# Views - Creation

68

Create a view for Employee-Pay

```
CREATE VIEW EMPPAY (EmpCode, NetPay,  
                    SalMonth) AS  
( SELECT    EmpCode,  
                  (Basic + Allow - Deduct), SalMonth  
            FROM    Salary );
```

# Use of Views

69

List employees and their ages

```
SELECT *  
FROM      EMPAGE;
```

# Views - Complex

69

Create a view for displaying the number of employees in each department

```
CREATE VIEW DeptEmpCount  
    (DeptCode, DeptEmpCount) AS  
    (SELECT    DeptCode, count(*)  
     FROM      Emp  
     GROUP BY  DeptCode);
```

# Use of Views

69

Create a view for display the total number of employees of the organization

```
CREATE VIEW EmpCount (EmpCount) AS  
  (SELECT count(*) as EmpCount  
   FROM      Emp);
```

# Use of Views

69

Display the percentage of employees in each department

```
select deptcode,(deptempcount/empcount)*100  
from deptempcount,empcount  
order by deptcode;
```

# Views - Updation

70

Update the view for Employee-Pay definition

```
alter view emppay(empcode,netpay,salmonth)
as (select empcode, (basic+allow-deduct)+1000,salmonth from salary);
```

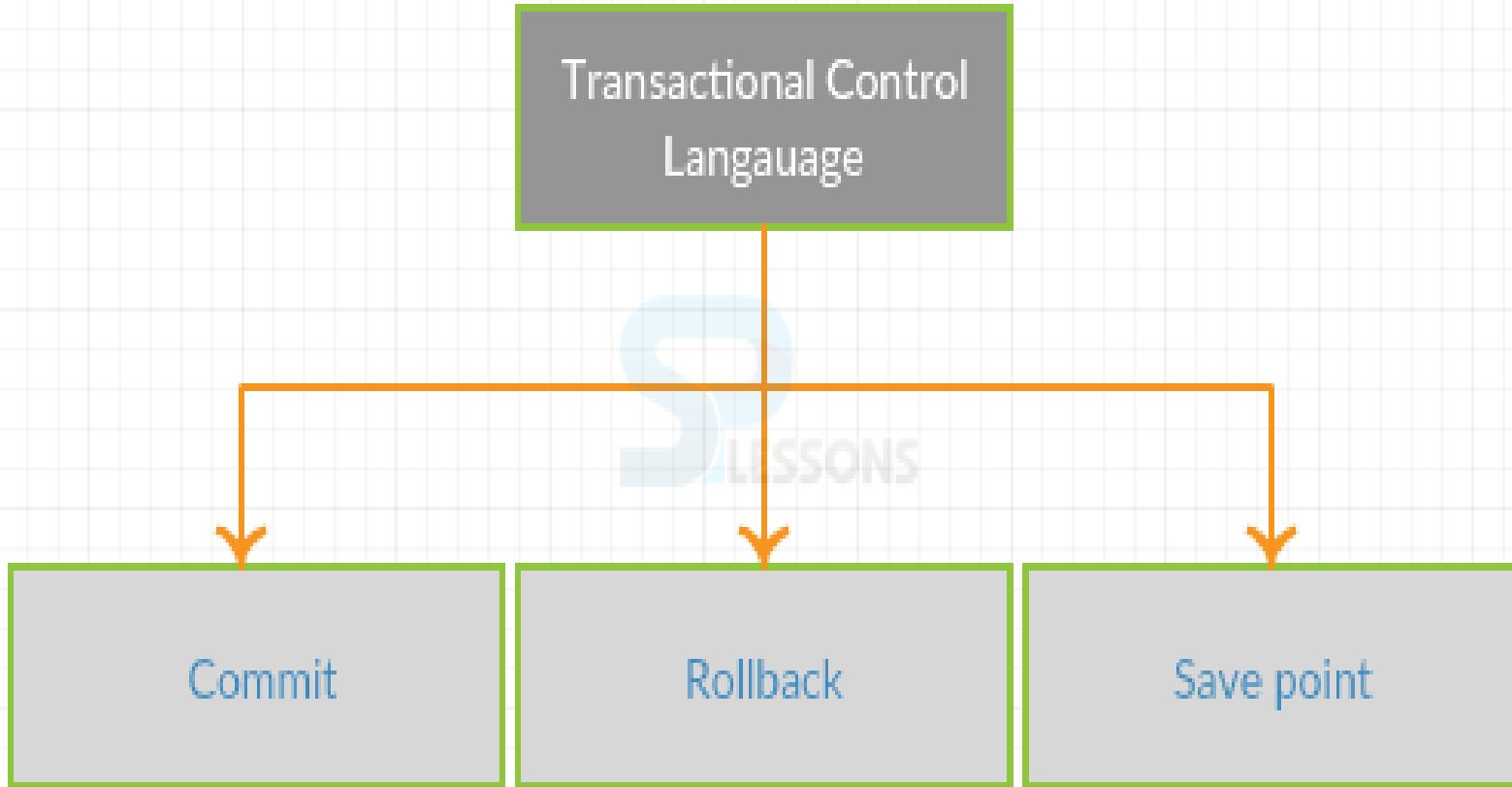
# Views - Deletion

**71**

**Delete the view Employee-Pay**

```
DROP VIEW EMPPAY;
```

# TCL Commands (Commit/Rollback/Savepoint)



# TCL Commands

- **Commit:**
  - Commit command is used to save all the transactions to the database.
- **Rollback:**
  - Rollback command is used to undo transactions that have not already been saved to the database.
- **SAVEPOINT:**
  - It is used to roll the transaction back to a certain point without rolling back the entire transaction.

# DCL Commands (GRANT/REVOKE/GRANT OPTION)

- The GRANT statement enables system administrators to grant privileges and roles, which can be granted to user accounts and roles.
- GRANT cannot mix granting both privileges and roles in the same statement. A given GRANT statement must grant either privileges or roles.

# MySQL – Error handling

# Contents

- Error Handling and Exceptions
- Types of Handler Actions
- How to write Handler
- Defining and handling exceptions in Stored Procedures and Functions

# MySQL Error Handling

- Handle error
  - continue
  - Or exit the current code block's execution, and issuing a meaningful error message.
- Handlers
  - Handle from general conditions such as warnings or exceptions
  - specific conditions e.g., specific error codes.

# Declaring a handler

- `DECLARE action HANDLER FOR condition_value statement;`
- The *action* accepts one of the following values:
  - CONTINUE : the execution of the enclosing code block ( BEGIN ... END ) continues.
  - EXIT : the execution of the enclosing code block, where the handler is declared, terminates.
- The *condition\_value* :
  - A MySQL error code.
  - A standard SQLSTATE value. Or it can be an SQLWARNING , NOTFOUND or SQLEXCEPTION condition, which is shorthand for the class of SQLSTATE values. The NOTFOUND condition is used for a [cursor](#) or SELECT INTO variable\_list statement.
  - A named condition associated with either a MySQL error code or SQLSTATE value.
- The *statement* could be a simple statement or a compound statement enclosing by the BEGIN and END keywords.

# MySQL handler precedence

In case you have multiple handlers that handle the same error, MySQL will call the most specific handler to handle the error first based on the following rules:

- An error always maps to a MySQL error code because in MySQL it is the most specific.
- An SQLSTATE may map to many MySQL error codes, therefore, it is less specific.
- An SQLEXCEPTION or an SQLWARNING is the shorthand for a class of SQLSTATES values so it is the most generic.
- Based on the handler precedence rules, MySQL error code handler, SQLSTATE handler and SQLEXCEPTION takes the first, second and third precedence.

# Named error condition

- `DECLARE condition_name CONDITION FOR condition_value;`
- The `condition_value` can be a MySQL error code such as `1146` or a `SQLSTATE` value.
- The `condition_value` is represented by the `condition_name`.
- After the declaration, you can refer to `condition_name` instead of `condition_value`



# NO SQL

# Contents

- Introduction to NoSQL database, Features of NoSQL Database
- Structured vs. Semi-structured and Unstructured Data
- Difference between RDBMS and NoSQL databases
- CAP Theorem, BASE Model
- Categories of NoSQL Databases: Key-Value Store, Document Store, Column-Oriented, Graph
- Introduction to MongoDB, Features of MongoDB
- MongoDB command interface and MongoDB compass

# Features- Not Only SQL

- No RDBMS
  - No relational
- Distributed Data Store
  - Horizontally scalable
- Schema-free / Flexible schema
  - Database JOINs generally not supported
- A huge amount of data
  - Eg Google/Facebook which collects terabits of data
- BASE properties

# Features- Not Only SQL

## **Vertical Scalability ( or Scale UP)**

- To add resources to a single node in a system, typically involving the addition of CPUs or memory to a single computer.
- Vertical scaling of existing systems enables them to use virtualization technology more effectively, as it provides more resources for the hosted set of operating system and application modules to share.
- Application scalability refers to the improved performance of running applications on a scaled-up version of the system

# Features- Not Only SQL

## **Horizontal Scalability (or scale out) :**

- Horizontal scaling means that you scale by adding more machines into your pool of resources.
- In a database world horizontal-scaling is often based on partitioning of the data
  - Each node contains only part of the data ,
  - In vertical-scaling the data resides on a single node and scaling is done through multi-core Spreading the load between the CPU and RAM resources of that machine.
- With horizontal-scaling it is often easier to scale dynamically by adding more machines into the existing pool
  - Vertical-scaling is often limited to the capacity of a single machine, scaling beyond that capacity often involves downtime and comes with an upper limit.

## Vertical “scalability”



## Horizontal scalability



Just scale it up

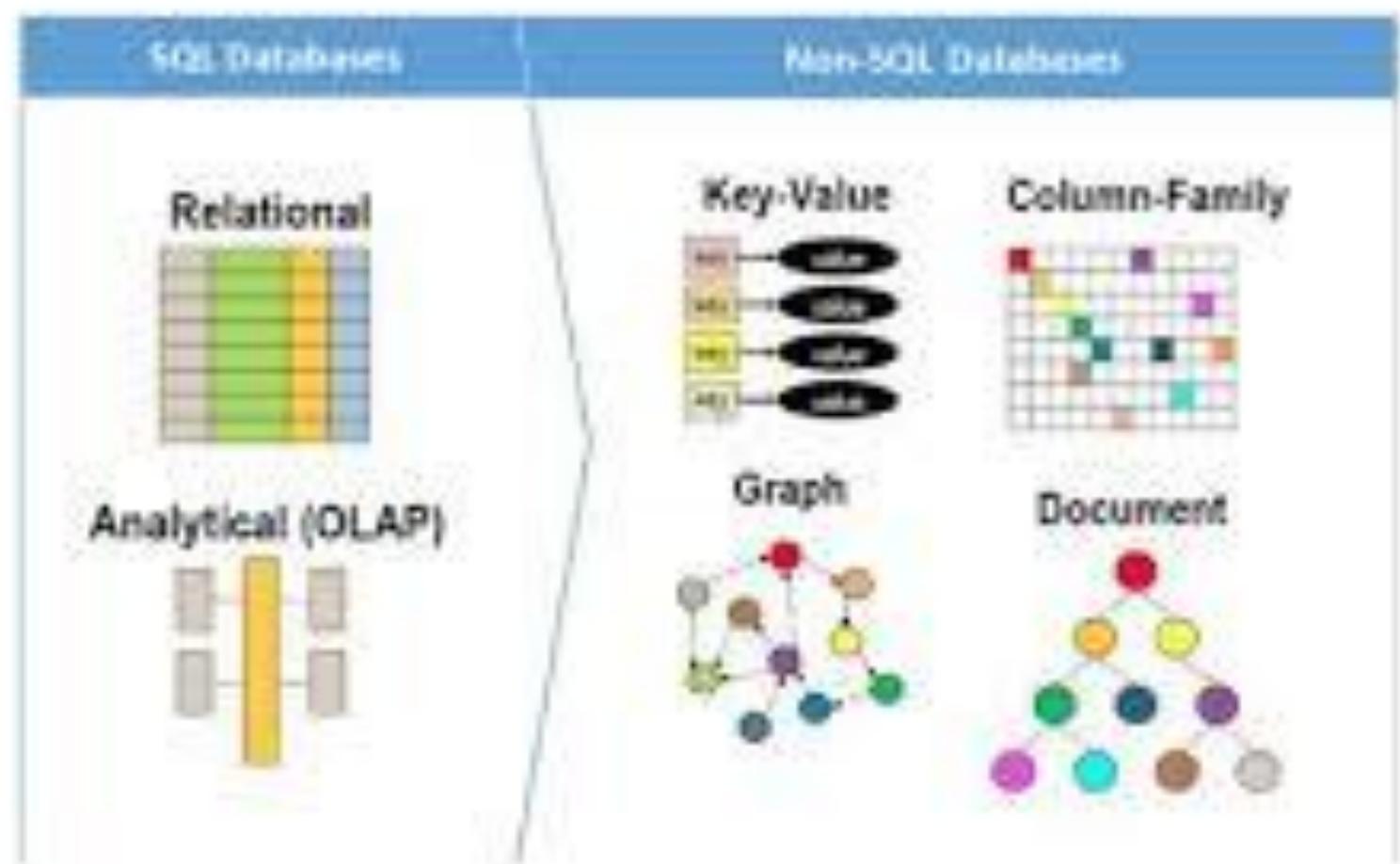
# NoSQL

- Provides a mechanism for
  - storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases
- Used in big data and real-time web applications
- NoSQL isn't a single product or technology, but an umbrella term for a category of databases

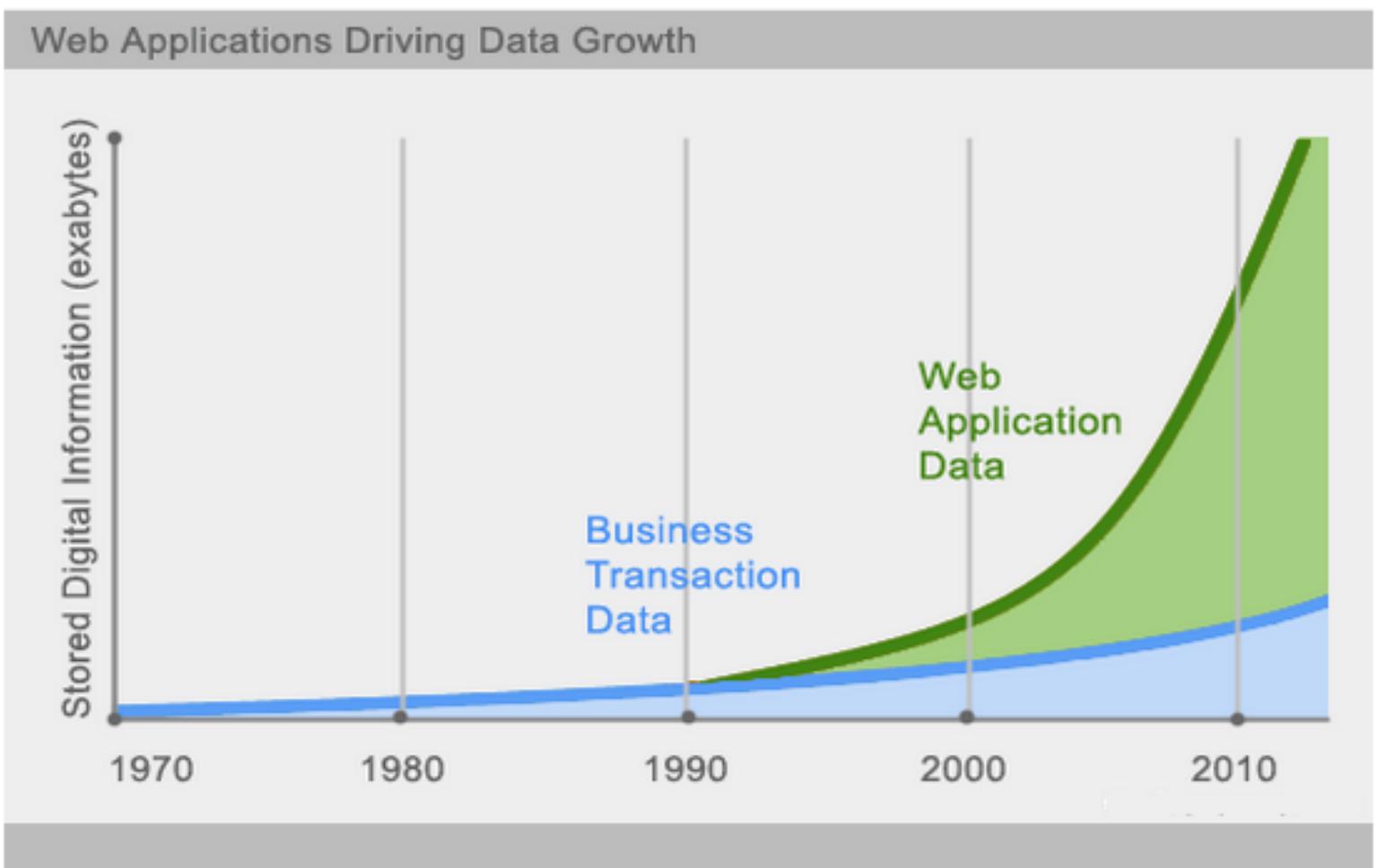
# NoSQL does not Provide

- Joins
- Group by
- ACID transactions
- SQL
- NoSQL databases reject:
  - Overhead of ACID transactions
  - “Complexity” of SQL
  - Burden of up-front schema design
  - Declarative query expression

# noSQL: “Not Only SQL”



# Requirement of NoSQL



# NoSQL - Users



# Structured Data

- Quantitative data
- Highly organized data
- structured query language (SQL) is the programming language used to manage structured data.
- By using a relational (SQL) database, business users can quickly input, search and manipulate structured data.
- Examples
  - dates, names, addresses, credit card numbers, etc.
  - Their benefits are tied to ease of use and access, while liabilities revolve around data inflexibility:

# Structured Data

## Pros

- Easily used by business users
- Accessible by more tools

## Cons

- Limited usage
  - Data with a predefined structure can only be used for its intended purpose, which limits its flexibility and usability.
- Limited storage options:
  - Structured data is generally stored in data storage systems with rigid schemas (e.g., “[data warehouses](#)”). Therefore, changes in data requirements necessitate an update of all structured data, which leads to a massive expenditure of time and resources.

# Structured Data

## Use cases for structured data

- **Customer relationship management (CRM):** CRM software runs structured data through analytical tools to create datasets that reveal customer behavior patterns and trends.
- **Online booking:** Hotel and ticket reservation data (e.g., dates, prices, destinations, etc.) fits the “rows and columns” format indicative of the pre-defined data model.
- **Accounting:** Accounting firms or departments use structured data to process and record financial transactions.

# Unstructured Data

- Qualitative data,
- Cannot be processed and analyzed via conventional data tools and methods.
- No predefined data model
- Managed in non-relational (NoSQL) databases.
- The importance of unstructured data is rapidly increasing.
- Examples
  - Text, mobile activity, social media posts, Internet of Things (IoT) sensor data, etc.
  - Their benefits involve advantages in format, speed and storage, while liabilities revolve around expertise and available resources

# Unstructured Data

## Pros

**Native format:** Unstructured data, stored in its native format, remains undefined until needed.

**Fast accumulation rates:** Since there is no need to predefine the data, it can be collected quickly and easily.

**Data lake storage:** Allows for massive storage and pay-as-you-use pricing, which cuts costs and eases scalability.

## Cons

**Requires expertise:** Due to its undefined/non-formatted nature, data science expertise is required to prepare and analyze unstructured data.

**Specialized tools:** Specialized tools are required to manipulate unstructured data, which limits product choices for data managers.

# Unstructured Data

## Use cases for unstructured data

- **Data mining:**

- Enables businesses to use unstructured data to identify consumer behavior, product sentiment, and purchasing patterns to better accommodate their customer base.

- **Predictive data analytics:**

- Alert businesses of important activity ahead of time so they can properly plan and accordingly adjust to significant market shifts.

- **Chatbots:**

- Perform text analysis to route customer questions to the appropriate answer sources.

# Semi -structured Data

- Semi-structured data (e.g., JSON, CSV, XML) is the “bridge” between structured and unstructured data.
  - It does not have a predefined data model and is more complex than structured data, yet easier to store than unstructured data.
  - Semi-structured data uses “metadata” (e.g., tags and semantic markers) to identify specific data characteristics and scale data into records and preset fields.
  - Metadata ultimately enables semi-structured data to be better cataloged, searched and analyzed than unstructured data.
- Semi-structured data vs. structured data:**
- A tab-delimited file containing customer data versus a database containing CRM tables.
- Semi-structured data vs. unstructured data:**
- A tab-delimited file versus a list of comments from a customer’s Instagram.

# CAP Theorem

- Three properties of a system
  - Consistency
    - All clients see the same data at the same time, no matter which node they connect to.
  - Availability
    - Any client making a request for data gets a response, even if one or more nodes are down. System can run even if parts have failed via replication
  - Partitions
    - The cluster must continue to work despite any number of communication breakdowns between nodes in the system.
- Distributed system can deliver **only two** of three desired characteristics

# CAP Theorem

- **CP database**

- Delivers consistency and partition tolerance at the expense of availability.
  - When a partition occurs between any two nodes, the system has to shut down the non-consistent node (i.e., make it unavailable) until the partition is resolved.

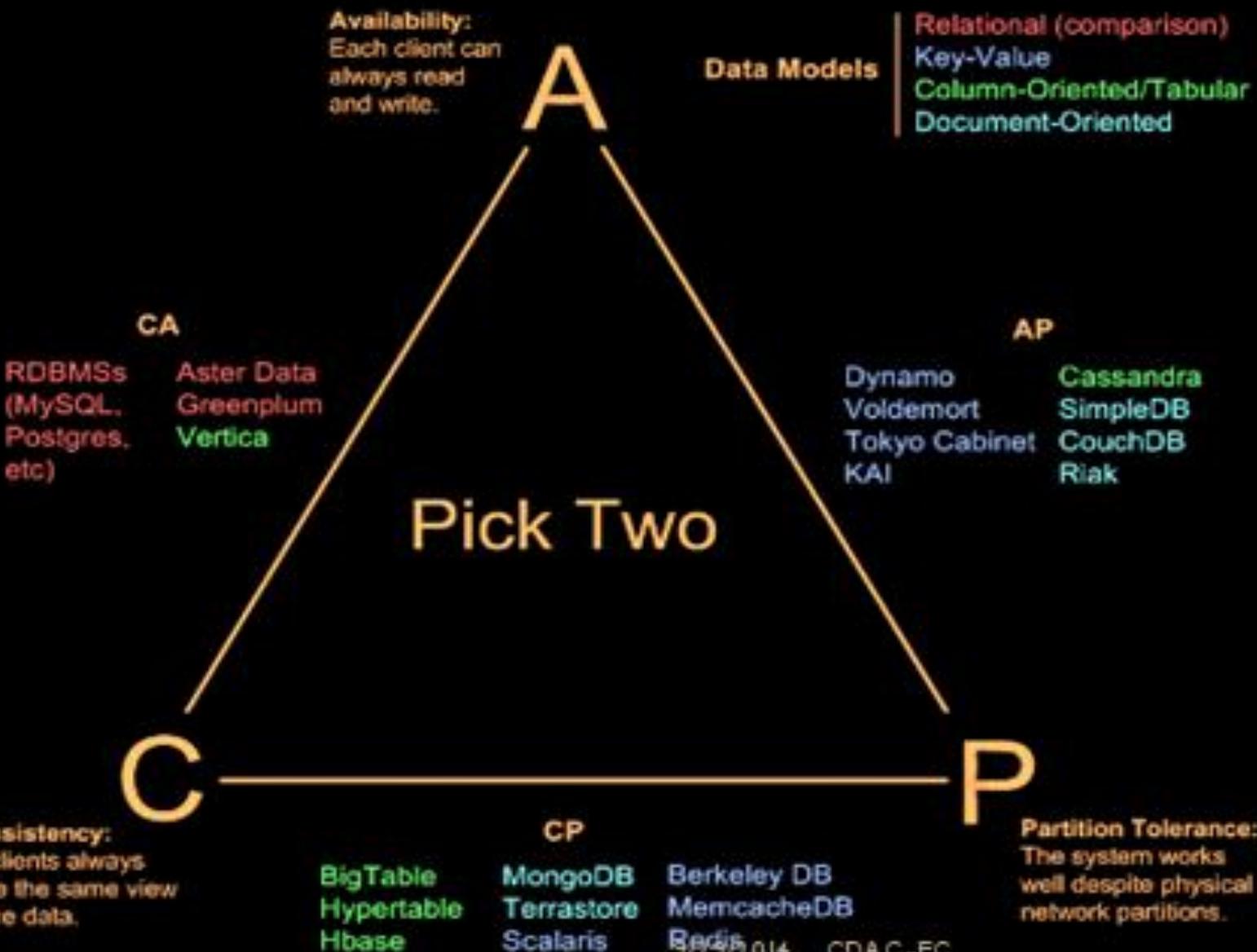
- **AP database:**

- An AP database delivers availability and partition tolerance at the expense of consistency.
  - When a partition occurs, all nodes remain available but those at the wrong end of a partition might return an older version of data than others.

- **CA database:**

- A CA database delivers consistency and availability across all nodes.
  - It can't do this if there is a partition between any two nodes in the system, however, and therefore can't deliver fault tolerance.

# CAP Theorem



# BASE Model

- **Basic Availability.**

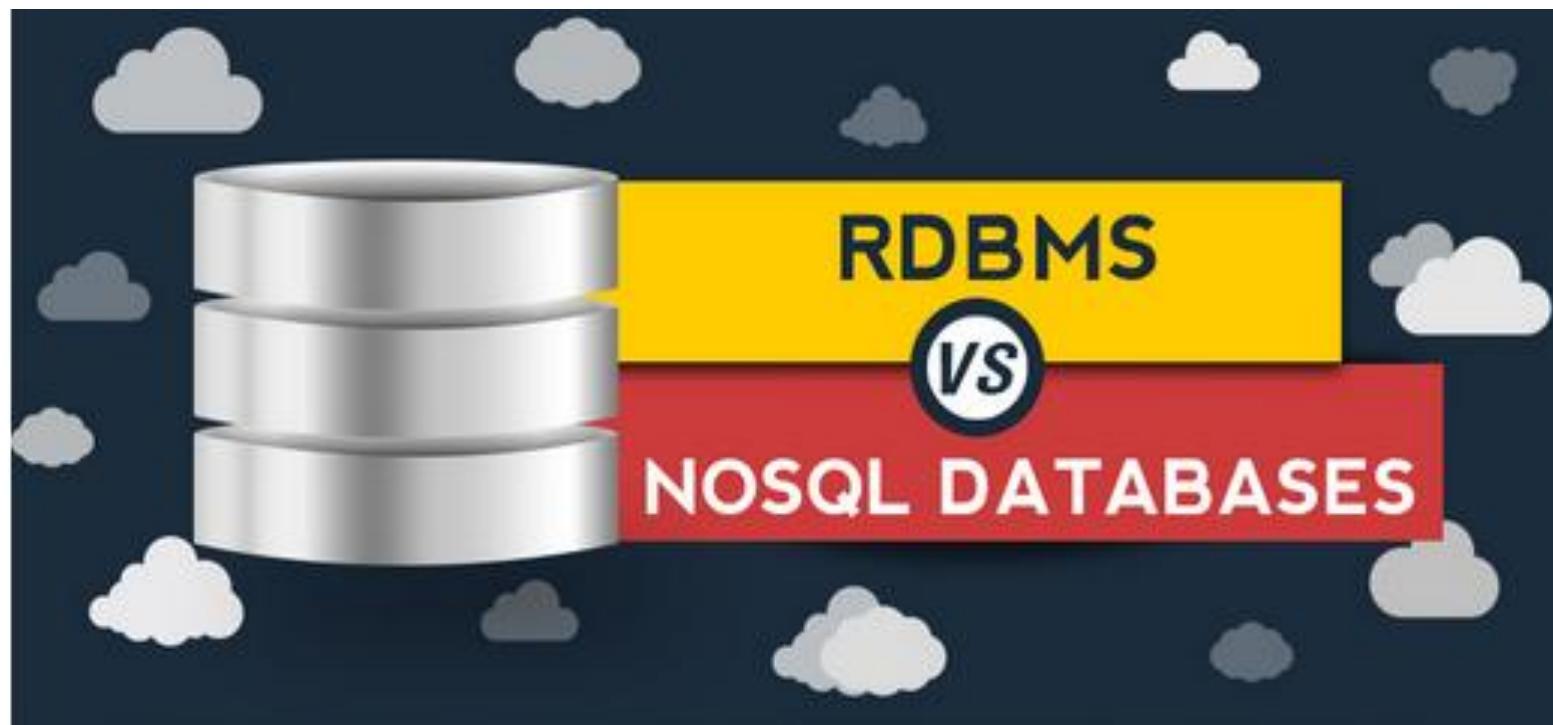
- Highly distributed approach to database management.
- NoSQL databases spread data across many storage systems with a high degree of replication.

- **Soft State.**

- BASE databases abandon the consistency requirements of the ACID model pretty much completely.
- One of the basic concepts behind BASE is that data consistency is the developer's problem and should not be handled by the database.

- **Eventual Consistency.**

- At some point in the future, data will converge to a consistent state
- That is a complete departure from the immediate consistency requirement of ACID that prohibits a transaction from executing until the prior transaction has completed and the database has converged to a consistent state.



# RDBMS VS NoSQL database

<u>RDBMS</u>	<u>NoSQL</u>
Structured and organized data	Stands for Not <u>Only SQL</u>
Structured query language (SQL)	No declarative query language
Data and its relationships are stored in separate tables.	No predefined schema
Data Manipulation Language, Data Definition Language	Variants - Key-Value Pair Store, Column Store, Document Store, Graph Store
Tight Consistency	Eventual consistency rather ACID property
ACID Transaction	CAP Theorem - Prioritizes high performance, high availability and scalability

*a*

Atomicity:  
Transactions  
are all or  
nothing

*c*

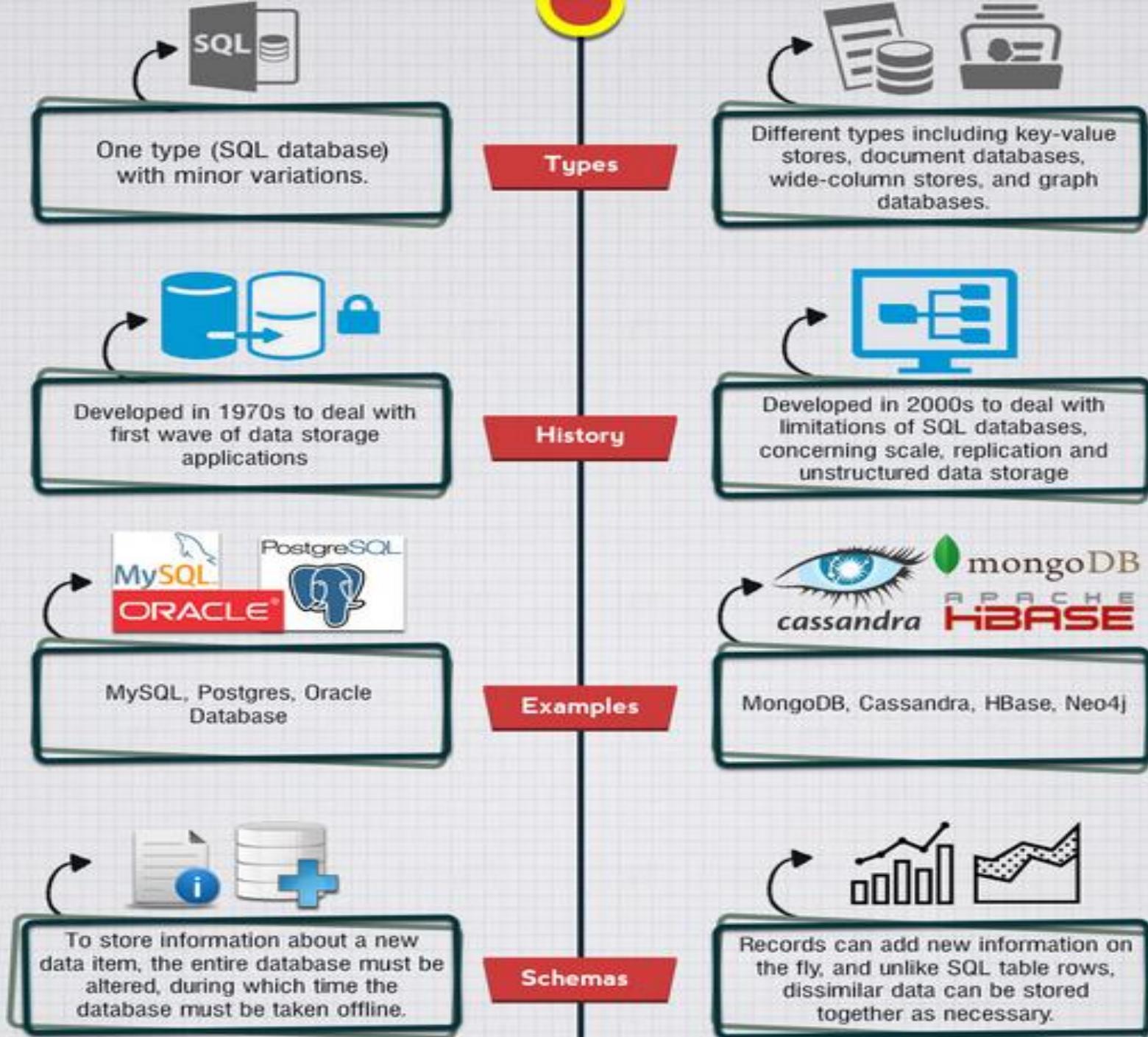
Consistency:  
Only valid data  
is saved

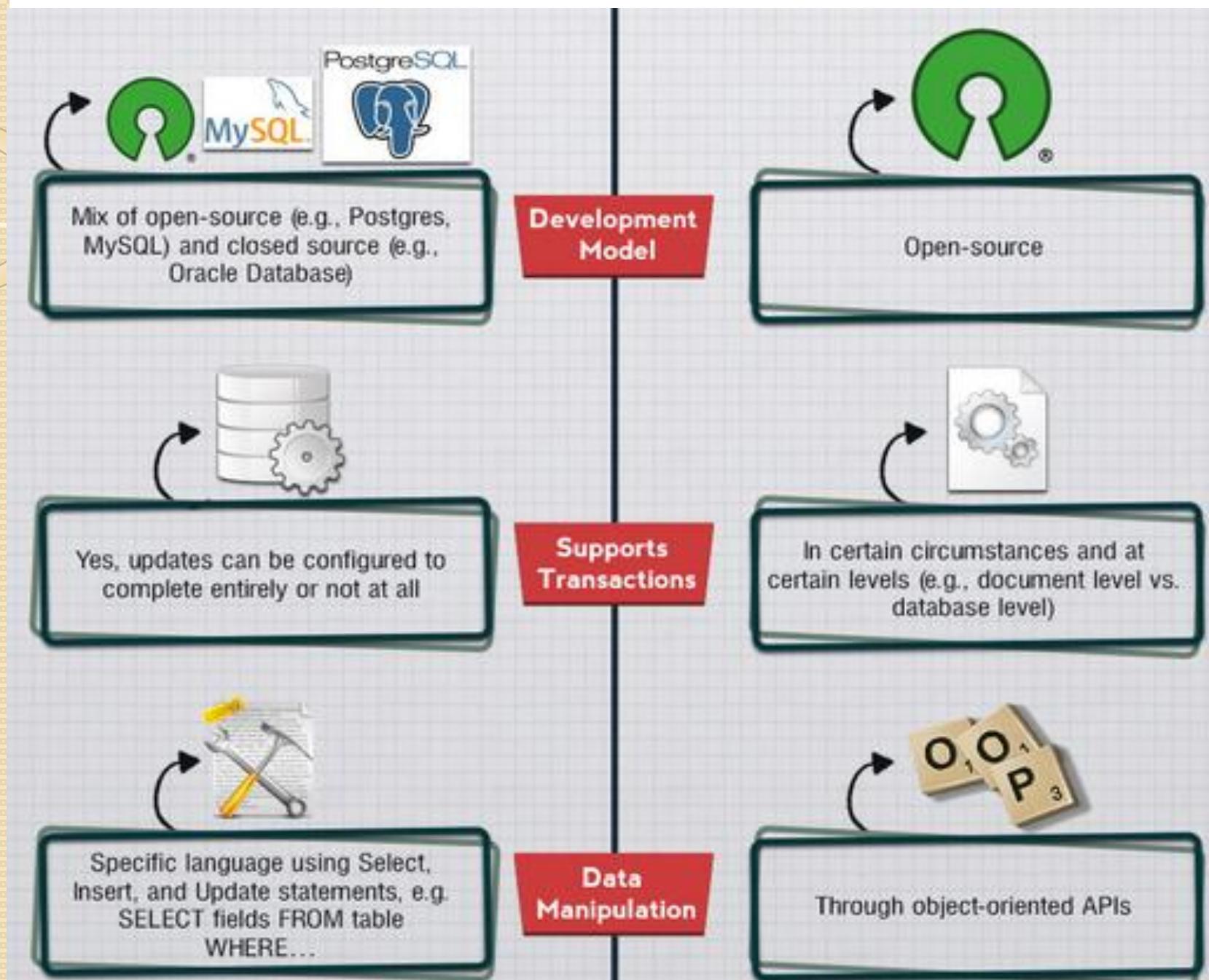
*i*

Isolation:  
Transactions  
do not affect  
each other

*d*

Durability:  
Written data  
will not be lost

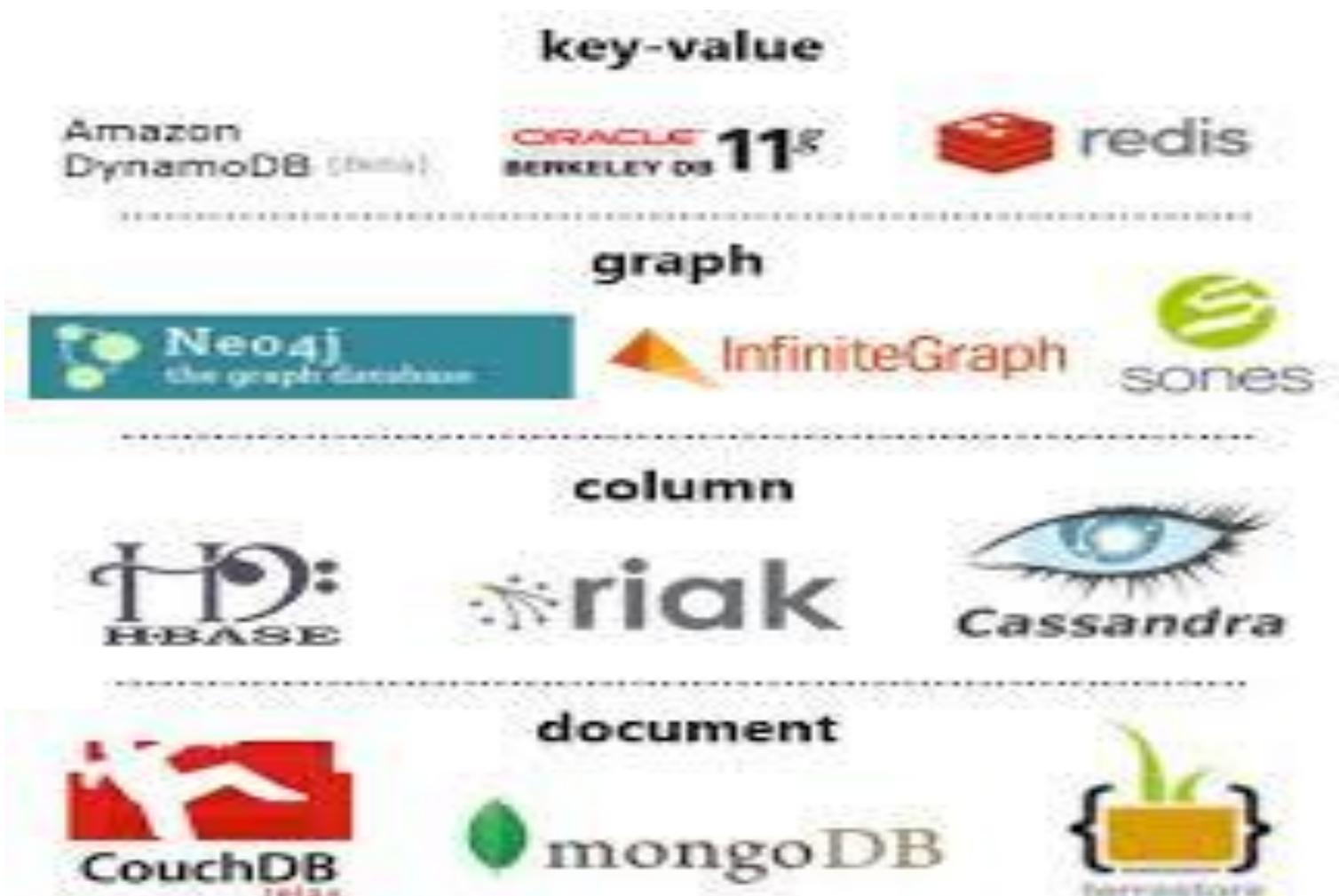




# Example –NoSQL Databases



# NoSQL Database Family



# Column-oriented databases

- Data tables are stored as sections of columns of data, rather than as rows of data.
- Column families are groups of related data that is often accessed together
- The basic unit of storage in Column-family databases is a column
- Example
  - Hadoop / Hbase
  - Cassandra :Apache Cassandra was initially developed at Facebook to power their Inbox Search feature
  - Cloudata :Google's Big table clone like HBase

# Column-Oriented Databases Cont

...

- The column is used as a store for the value, and has a timestamp that is used to differentiate the valid content from stale ones.
- Application will use the timestamp to find out which of the stored values in the backup nodes are up-to-date.
- Column Family
  - A container for columns, analogous to table in a relational database.
  - The column Family has a name, a map with a key and a value(which is a map containing columns).

# Example

- Cassandra
- Hbase
- Hypertable
- Amazon Simple DB

# Row Vs Column

RowId	Empld	Lastname	Firstname	Salary
001	10	Smith	Joe	40000
002	12	Jones	Mary	50000
003	11	Johnson	Cathy	44000
004	22	Jones	Bob	55000

- Row – oriented database

001:10,Smith,Joe,40000;002:12,Jones,Mary,50000;003:11,Johnson,Cathy,44000;004:22,J

- Column oriented database

10:001,12:002,11:003,22:004;Smith:001,Jones:002,Johnson:003,Jones:004;Joe:001,Mary  
50000:002,44000:003,55000:004;

# Row Vs Column

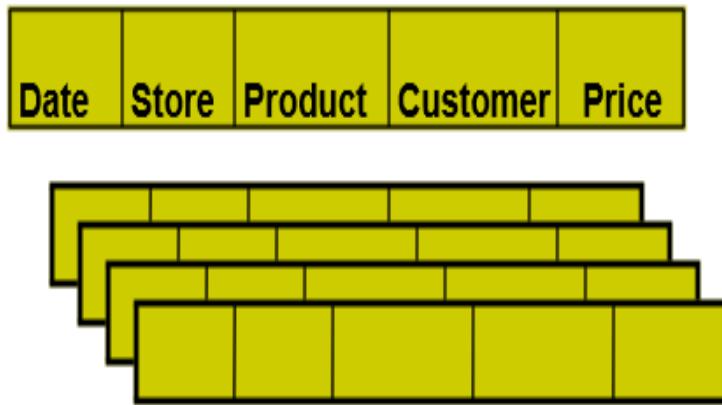
- As two of the records store the same value, "Jones", it is possible to store this only once in the column store, along with pointers to all of the rows that match it.
- For many common searches, like "find all the people with the last name Jones", the answer is retrieved in a single operation.
- Other operations, like counting the number of matching records or performing math over a set of data, can be greatly improved through this organization.

# Contrasting Column Databases with RDBMS

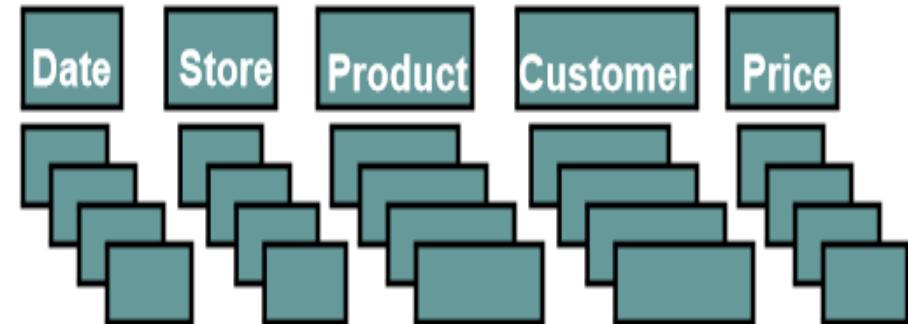
- Column-oriented database
  - minimal need for schema definition
  - easily accommodate newer columns
  - predefined column-family
    - set of columns grouped together into a bundle
  - Column family (no data type) - column in an RDBMS (with data type)
  - Column databases designed to scale and can easily accommodate millions of columns and billions of rows
- In a row-oriented indexed system, the primary key is the rowid that is mapped to indexed data.
  - In the column-oriented system, the primary key is the data, mapping back to rowids.

# Contrasting Column Databases with RDBMS Cont ...

**row-store**



**column-store**



+ easy to add/modify a record

- might read in unnecessary data

+ only need to read in relevant data

- tuple writes require multiple accesses

=> *suitable for read-mostly, read-intensive, large data repositories*

# HBase users



# Document Model

- Pair each key with a complex data structure known as a document.
- Helpful for modeling unstructured and polymorphic data.
- It also makes it easier to evolve an application during development , such as adding new fields.
- Data can be queried based on any fields in a document

- Traditional
  - schema design focus on data storage
  - What answer do we have
- Document store
  - Focus on data usage
  - What question do I have

# DOCUMENT STORE

- Documents are grouped together into collections
- Collections - relational tables.
- Collections don't impose strict schema constraints
  - Records are not documents in the sense of a word processing document
- Structure of any document can be modified
  - By adding and removing members from the document - by reading the document into program, modifying it and re-saving it
  - By using various update commands.

# DOCUMENT STORE

- Records within a single table can have different structures.
- An example record from Mongo, using JSON format, might look like

```
{  
  "_id" : ObjectId("4fccbf281168a6aa3c215443"),  
  "first_name" : "Thomas",  
  "last_name" : "Jefferson",  
  "address" : {  
    "street" : "1600 Pennsylvania Ave NW",  
    "city" : "Washington",  
    "state" : "DC"  
  }  
}
```

Embedded  
document

# Document Store - Internals

- Document Stores

- Like Key-Value Stores, except Value is a “Document”

- Data model: (key, “document”) pairs

- Basic operations: I

- Insert (key, document),
  - Fetch(key), Update(key),
  - Delete(key)

- Also Fetch() based on document contents

- Example systems

- CouchDB, MongoDB

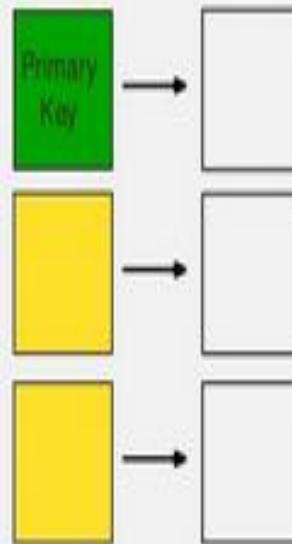
- Document stores

- Store arbitrary/extensible structures as a “value”

# Relational Vs Document

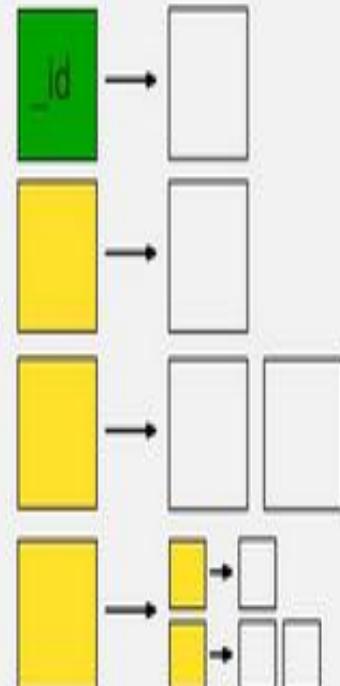
## Relational

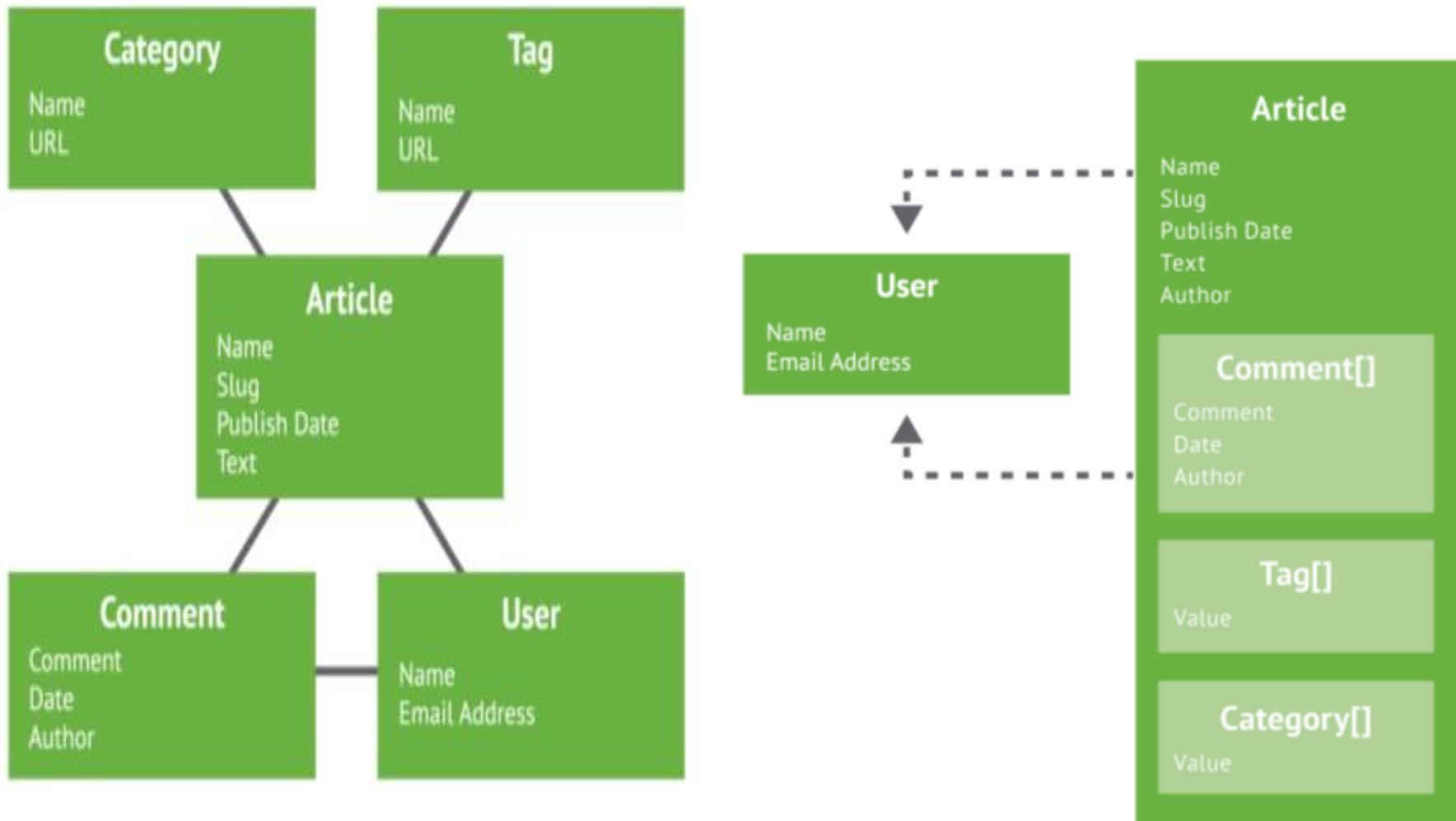
- Two-dimensional storage (tuples)
- Each field contains a single value
- Query on **any field**
- Very structured schema (table)
- In-place updates
- Normalization process requires many tables, joins, **indexes**, and poor data locality



## Document

- N-dimensional storage
- Each field can contain 0, 1, many, or embedded values
- Query on **any field & level**
- **Flexible schema**
- Inline updates \*
- Embedding related data has optimal data locality, requires fewer **indexes**, has better **performance**





# Advantages of the Document Model

- More natural to represent data at the database level
- An aggregated document can be accessed with a single call to the database
  - rather than having to JOIN multiple tables to respond to a query.
- The MongoDB document is physically stored as a single object, requiring only a single read from memory or disk.
  - RDBMS JOINs require multiple reads from multiple physical locations.
- Distributing the database across multiple nodes (a process called sharding) is easier
  - horizontal scalability

# DOCUMENT STORE

- “Documents” are encoded in a standard data exchange format
  - XML, JSON (JavaScript Object Notation) or BSON (Binary JSON).
- Unlike the simple key-value stores, the value column in document databases contains semi-structured data
  - specifically attribute name/value pairs.
- A single column can house hundreds of such attributes
- Number and type of attributes recorded can vary from row to row.
- Both keys and values are fully searchable in document databases.

# DOCUMENT STORE

- Each document is stored in BSON format.
- BSON is a binary-encoded representation of a JSON-type document format
  - nested set of key/value pairs.
  - JSON – JavaScript Object Notation
- BSON is a superset of JSON
  - supports additional types
    - regular expression,
    - binary data,
    - date.
- Each document has a unique identifier, which MongoDB can generate like auto-generated object ids

# *Transform Your World*

## Drivers



## Documents

### BSON

### dot notation

```
db.persons.find( { "address.state": "NY" } )  
db.persons.ensureIndex( { "address.zipcode": 1 } )  
  { "firstName": "Elliot",  
    "lastName": "Horowitz",  
    "address": { "city": "New York",  
                "state": "NY",  
                "zipcode": "10021" }  
  },  
  { "phoneNumber": {  
    "type": "fax",  
    "number": "646-619-4253" } }
```

### Binary JSON

MongoDB Wire Protocol

## Databases

Storage  
Performance  
Querying  
Indexes  
Replication  
Sharding  
Updating  
Aggregation  
MapReduce  
GridFS  
Geospatial

*Data Impedance Matching*

# MongoDB- Features

- MongoDB provides high performance data persistence.
  - Support for embedded data models reduces I/O activity on database system.
  - Indexes support faster queries and can include keys from embedded documents and arrays.
- High Availability
  - A replica set is a group of MongoDB servers that maintain the same data set, providing redundancy and increasing data availability.
- Automatic Scaling
  - MongoDB provides horizontal scalability as part of its core functionality.
  - Automatic sharding distributes data across a cluster of machines.
  - Replica sets can provide eventually-consistent reads for low-latency high throughput deployments.

# Distributed Key-Value Systems

- Key-Value Pair (KVP) Stores
  - Access data (values) by strings called keys.
  - Data has no required format – data may have any format
  - Extremely simple interface
- Data model: (key, value) pairs
- NoSQL Key-Value store is a single table with two columns:
  - one being the (Primary) Key, and the other being the Value.
- Basic Operations: Insert (key, value), Fetch (key), Update (key), Delete (key)
  - Implementation: efficiency, scalability, fault-tolerance
  - Uses a hash table in which there exists a unique key and a pointer to a particular item of data

# Example- Key Value

- Riak
- Redis
- Memcached DB
- Berkeley DB
- Hamster DB (especially suited for embedded use)
- Amazon Dynamo DB (not open source)
- Project Voldemort (Open Source Implementation of Dynamo DB)

# When to use NoSQL Databases

## Column Families/Wide Column Stores

- Usage: Read/write intensive applications (Ex: Social Networking)
- Popular Databases: HBase, Cassandra

## Document Store

- Usage: Working with occasionally changing/consistent data (Ex: CRM Systems)
- Popular Databases: CouchDB, MongoDB

## NO SQL

### Databases

## Graph Databases

- Usage: Spatial Data storage (Ex: Geographical Information Systems)
- Popular Databases: Neo4J, Bigdata

## Key Value/ Tulip Store

- Usage: Briskly changing data and high availability (Ex: Stock quotes and prices)
- Popular Databases: Riak, Redis, Azure Table Storage

# When to use NoSQL Databases

Key / Value Based	Column Based	Document Based	Graph Based
<p><b>Caching:</b> Quickly storing data for sometimes frequent - future use.</p> <p><b>Queue-ing:</b> Some K/V stores (e.g. Redis) supports lists, sets, queues and more.</p> <p><b>Keeping live information:</b></p>	<p><b>Keeping unstructured, non-volatile information:</b> If values needs to be kept for long periods of time</p> <p><b>Scaling</b> Column based data stores are highly scalable by nature.</p>	<p><b>Nested information:</b> Document-based data stores allow you to work with deeply nested, complex data structures.</p> <p><b>JavaScript friendly:</b> the way they interface with applications: Using JS friendly JSON.</p>	<p><b>Handling complex relational information:</b> make it extremely efficient and easy to use to deal with complex but relational information, such as the connections between two entities and various degrees of other entities indirectly related to them.</p> <p><b>Modelling and handling classifications</b></p>

# Query Operations in NoSQL Databases

Details – MongoDB

# Contents

- ▶ Background - Document Store
- ▶ Introduction to MongoDB (NoSQL)
- ▶ Performing CRUD Operations
- ▶ Creating Records
- ▶ Accessing Data
- ▶ Updating and Deleting Data
- ▶ Querying MongoDB
- ▶ Similarities Between SQL and MongoDB Query Features

# Document Model

- Pair each key with a complex data structure known as a document.
- Helpful for modeling unstructured and polymorphic data.
- It also makes it easier to evolve an application during development, such as adding new fields.
- Data can be queried based on any fields in a document

# DOCUMENT STORE

- Document databases –
  - Good for storing and managing Big Data-size collections of literal documents
    - like text documents, email messages, and XML documents
    - conceptual “documents” like de-normalized (aggregate) representations of a database entity
- Good for storing “sparse” data
  - irregular (semi-structured) data that would require an extensive use of “nulls” in an RDBMS.

# DOCUMENT STORE

- “Documents” are encoded in a standard data exchange format
  - XML, JSON (JavaScript Object Notation) or BSON (Binary JSON).
- Unlike the simple key-value stores, the value column in document databases contains semi-structured data
  - specifically attribute name/value pairs.
- A single column can house hundreds of such attributes
- Number and type of attributes recorded can vary from row to row.
- Both keys and values are fully searchable in document databases.

# DOCUMENT STORE

- Each document is stored in BSON format.
- BSON is a binary-encoded representation of a JSON-type document format
  - nested set of key/value pairs.
  - JSON – JavaScript Object Notation
- BSON is a superset of JSON
  - supports additional types
    - regular expression,
    - binary data,
    - date.
- Each document has a unique identifier, which MongoDB can generate like auto-generated object ids

# Advantages of the Document Model

- More natural to represent data at the database level
- An aggregated document can be accessed with a single call to the database
  - rather than having to JOIN multiple tables to respond to a query.
- The MongoDB document is physically stored as a single object, requiring only a single read from memory or disk.
  - RDBMS JOINs require multiple reads from multiple physical locations.
- Distributing the database across multiple nodes (a process called sharding) is easier
  - horizontal scalability

# Introduction to MongoDB

- ▶ MongoDB - document oriented data base from the NoSQL database family.
- ▶ Suitable for
  - Storing and managing big data-size collections like text documents, email messages, and XML documents etc.
- ▶ Dynamic schema
  - Unlike predefined schema in RDBMS

# MongoDB- Features

- High Performance
  - Support for embedded data models
    - reduces I/O activity on database system.
  - Indexes support
    - faster queries and can include keys from embedded documents and arrays.
- High Availability
  - A replica set is a group of MongoDB servers that maintain the same data set
    - provide redundancy and increased data availability.

# MongoDB- Features

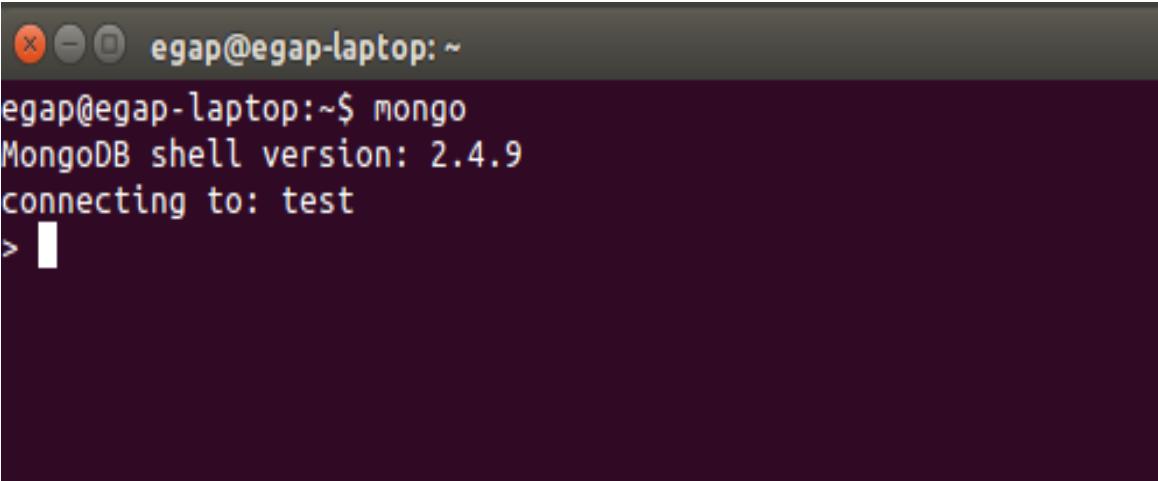
- Automatic Scaling
  - MongoDB provides horizontal scalability as part of its core functionality.
  - Automatic sharding distributes data across a cluster of machines.
  - Replica sets can provide eventually-consistent reads for low-latency high throughput deployments.

## ▶ Mongo DB editions

- **mongo Shell Edition**
- Node.JS Edition
- Python Edition
- C++ Edition
- Java Edition
- C# Edition

- ▶ MongoDB includes the following tools in its bin folder:
  - Mongod
    - database server
  - Mongo
    - database client
  - Mongodump
    - dumps the database to a BSON file (Binary JSON)
  - Mongorestore
    - rebuilds the MongoDB from the BSON file
  - Mongoexport
    - exports the MongoDB database to a text format, which is either JSON or CSV
  - Mongoimport
    - imports the MongoDB database from the JSON or CSV file

# MongoDB Shell



```
egap@egap-laptop:~$ mongo
MongoDB shell version: 2.4.9
connecting to: test
> |
```

- ▶ Shell connects to a MongoDB server on startup
- ▶ It is a JavaScript Interpreter, capable of running Java Script program.
- ▶ Standalone MongoDB client

```
surabhi@surabhi-seng:~$ mongo
MongoDB shell version: 2.6.4
connecting to: test
Server has startup warnings:
2014-09-05T09:42:06.080+0530 [initandlisten]
2014-09-05T09:42:06.080+0530 [initandlisten] ** NOTE: This is a 32 bit MongoDB b
inary.
2014-09-05T09:42:06.080+0530 [initandlisten] **      32 bit builds are limited
to less than 2GB of data (or less with --journal).
2014-09-05T09:42:06.080+0530 [initandlisten] **      Note that journaling defau
lts to off for 32 bit and is currently off.
2014-09-05T09:42:06.080+0530 [initandlisten] **      See http://dochub.mongodb.org/core/32bit.
2014-09-05T09:42:06.080+0530 [initandlisten]
> db
test                                     → Default Database
> show dbs
admin (empty)
local 0.078GB
> use mydb_course                         → Create a new Database
switched to db mydb_course
> db
mydb_course
```

# MongoDB- Documents

```
{  
    "_id" : ObjectId("54c955492b7c8eb21818bd09"),  
    "address" : {  
        "street" : "2 Avenue",  
        "zipcode" : "10075",  
        "building" : "1480",  
        "coord" : [ -73.9557413, 40.7720266 ],  
    },  
    "borough" : "Manhattan",  
    "cuisine" : "Italian",  
    "grades" : [  
        {  
            "date" : ISODate("2014-10-01T00:00:00Z"),  
            "grade" : "A",  
            "score" : 11  
        },  
        {  
            "date" : ISODate("2014-01-16T00:00:00Z"),  
            "grade" : "B",  
            "score" : 17  
        }  
    "name" : "Vella",  
    "restaurant_id" : "41704620"  
}
```

# SQL & MongoDB

SQL Term	MongoDB Term
database (schema)	database
table	collection
index	index
row	document
column	field
joining	linking & embedding
partition	shard

```

> j={ name : "mongo" }
[ "name" : "mongo" ]
> k={ x : 3 }
[ "x" : 3 ]
> db.DBTcourse.insert(j)
writeResult({ "nInserted" : 1 })
> db.DBTcourse.insert(k)
writeResult({ "nInserted" : 1 })
> show collections
DBTcourse
system.indexes
> db.DBTcourse.find()
{ "_id" : ObjectId("54095d1bb60fece55ee2b96b"), "name" : "mongo" }
{ "_id" : ObjectId("54095d20b60fece55ee2b96c"), "x" : 3 }

```

two documents named j and k

Insert the j and k documents into the “DBTcourse”collection

Confirms the document exists in collection

# Configure and access the store using the Admin Console

- ▶ MongoDB will not permanently create a database until you insert data into that database
- ▶ MongoDB will create a collection implicitly upon its first use.
- ▶ Need not to create a collection before inserting data.
- ▶ MongoDB uses dynamic schemas
  - Need not to specify the structure of documents before inserting them into the collection.

# \_id and ObjectIds

- ▶ In a single collection every document must have unique value for \_id
- ▶ ObjectId is default type for “\_id”
- ▶ ObjectId uses 12-byte of storage (24 hexadecimal digits)
  - a 4-byte value representing the seconds
  - a 3-byte machine identifier,
    - Different machine will not generate colliding ObjectIDs
  - a 2-byte process id, and
  - a 3-byte counter, starting with a random value.
- ▶ ObjectId("56cab47a407e1684708f1a5c")
- ▶ First 9 bytes ensures uniqueness across machine and process
- ▶ Last 3 bytes are incremental counter, responsible for uniqueness within second in a single process.

# Query Interface

The following diagram highlights the components of a MongoDB query operation:

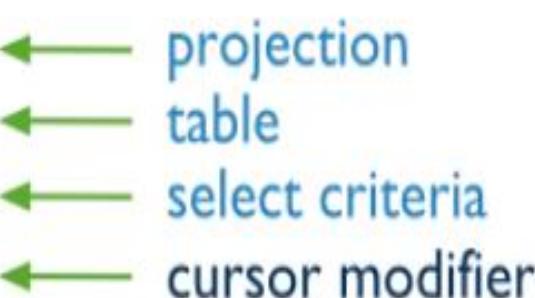
```
db.users.find(  
    { age: { $gt: 18 } },  
    { name: 1, address: 1 }  
).limit(5)
```



The components of a MongoDB find operation.

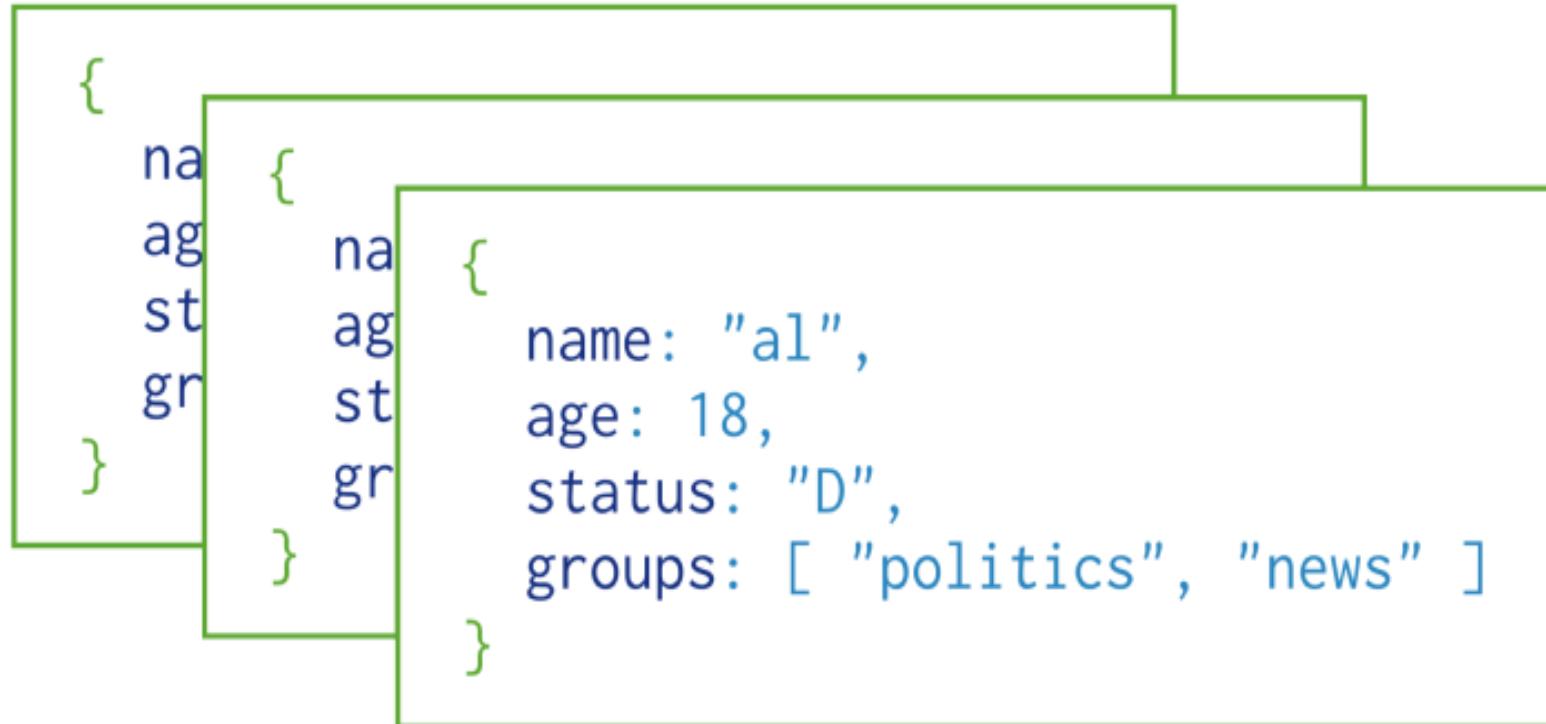
The next diagram shows the same query in SQL:

```
SELECT _id, name, address  
FROM users  
WHERE age > 18  
LIMIT 5
```



The components of a SQL SELECT statement.

# Database Operations



Collection

# CRUD – Read Operation

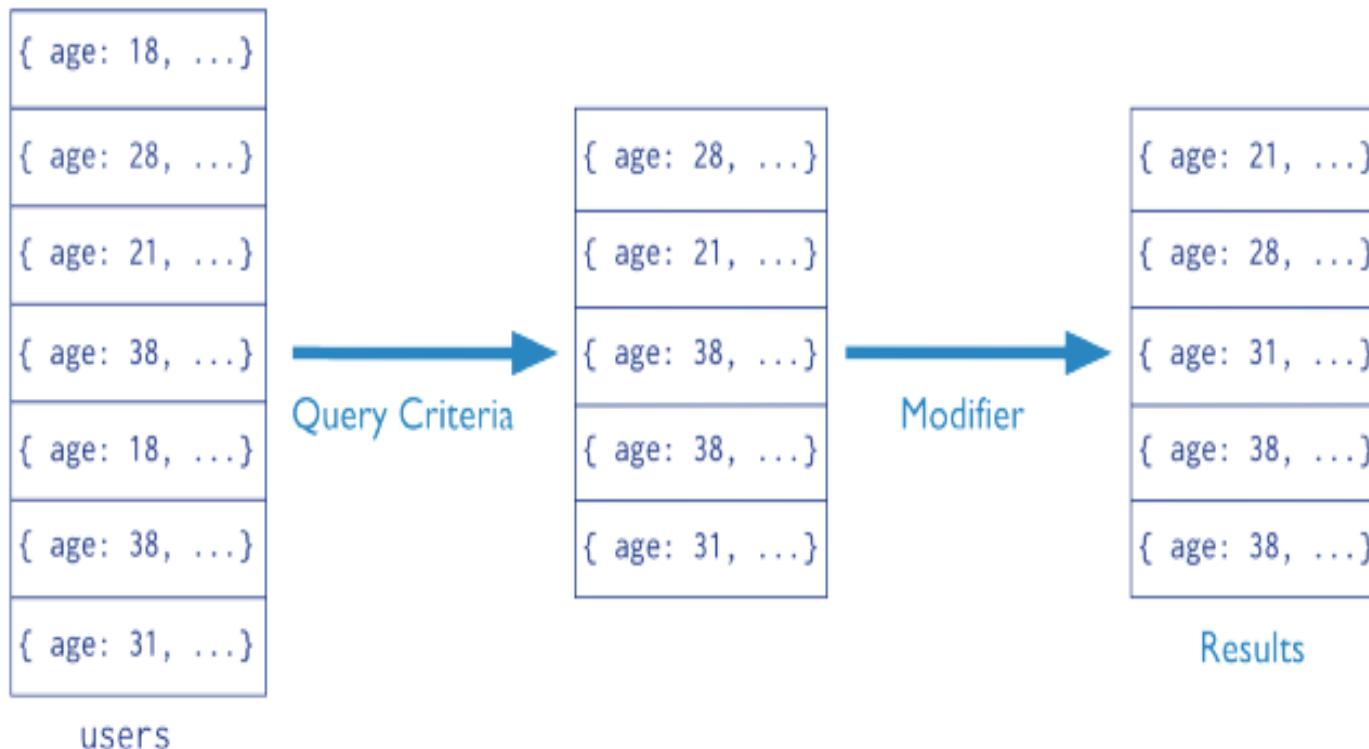


## ▶ Read Operations :

- Operations that select and return documents to clients, including the query specifications.
  - Cursors
    - Queries return iterable objects, called cursors, that hold the full result set.
  - Query Optimization :
    - Analyze and improve query performance.
    - Create an Index to Support Read Operations
  - Distributed Queries :
    - Describes how **sharded** clusters and replica sets affect the performance of read operations.

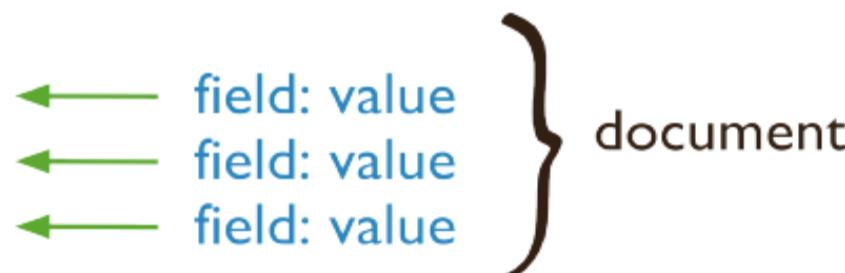
# CRUD – Read Operation – Example

Collection                    Query Criteria                    Modifier  
`db.users.find( { age: { $gt: 18 } } ).sort( {age: 1} )`

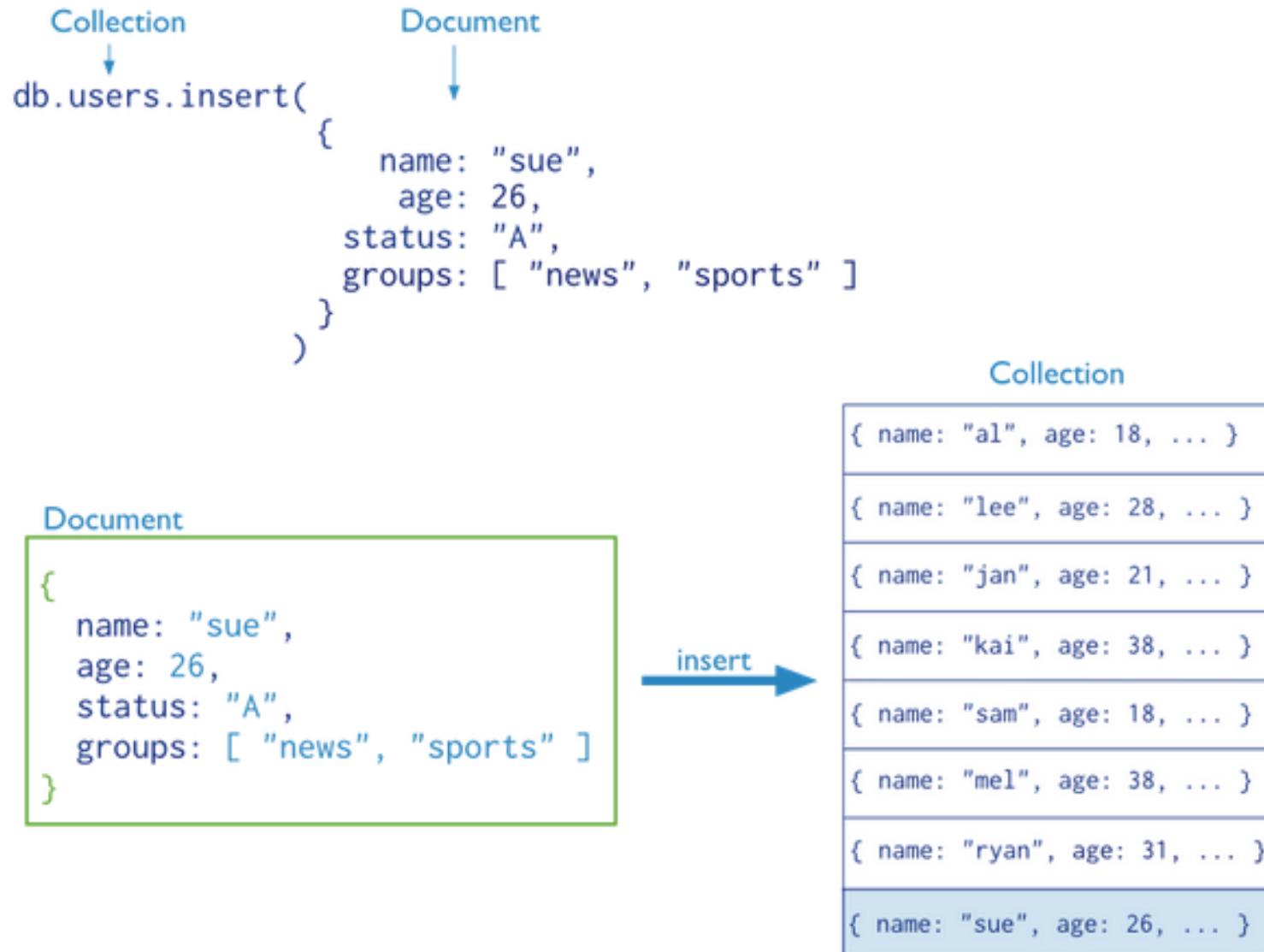


# CRUD- Write Operations

```
db.users.insert ( ← collection
  {
    name: "sue", ← field: value
    age: 26, ← field: value
    status: "A" ← field: value
  }
)
```



# CRUD – Write Operation – Example



# Insert an Array of Documents

```
> var mydocuments =  
...   [  
...     {  
...       item: "ABC2",  
...       details: { model: "14Q3", manufacturer: "M1 Corporation" },  
...       stock: [ { size: "M", qty: 50 } ],  
...       category: "clothing"  
...     },  
...     {  
...       item: "MNO2",  
...       details: { model: "14Q3", manufacturer: "ABC Company" },  
...       stock: [ { size: "S", qty: 5 }, { size: "M", qty: 5 }, { size: "L",  
qty: 1 } ],  
...       category: "clothing"  
...     },  
...     {  
...       item: "IJK2",  
...       details: { model: "14Q2", manufacturer: "M5 Corporation" },  
...       stock: [ { size: "S", qty: 5 }, { size: "L", qty: 1 } ],  
...       category: "houseware"  
...     }  
...   ];  
> db.inventory.insert( mydocuments );  
BulkWriteResult({  
  "writeErrors" : [ ],  
  "writeConcernErrors" : [ ],  
  "nInserted" : 3,  
  "nUpserted" : 0,  
  "nMatched" : 0,  
  "nModified" : 0,  
  "nRemoved" : 0,  
  "upserted" : [ ]  
})
```

# CURD: Creating Records in a Document-Centric Database

```
{  
    order_date: new Date(),  
    "line_items": [  
        {  
            item : {  
                name: "latte",  
                unit_price: 4.00  
            },  
            quantity: 1  
        },  
        {  
            item: {  
                name: "cappuccino",  
                unit_price: 4.25  
            },  
            quantity: 1  
        },  
        {  
            item: {  
                name: "regular",  
                unit_price: 2.00  
            },  
            quantity: 2  
        }  
    ]  
}
```

# CRUD – Update and Delete

- ▶ **Update :**
  - db.collection.update() and the db.collection.save() methods method updates a single document by default.
  - with the multi option, update() can update all documents in a collection that match a query.

```
db.users.update(  
  { age: { $gt: 18 } },  
  { $set: { status: "A" } },  
  { multi: true }  
)
```
- ▶ **Delete :**
  - remove documents from a collection
  - db.collection.remove() method
    - accepts a query criteria to determine which documents to remove
    - Example

```
db.users.remove(  
  { status: "D" }  
)
```
- ▶ To remove all documents from a collection, pass an empty query document {} to the remove() method.

# Querying NoSQL Stores –MongoDB

## ▶ Specify Equality Condition

- use the query document { <field>: <value> } to select all documents that contain the <field> with the specified <value>.

```
> db.inventory.find( { category: "houseware" } )
{ "_id" : ObjectId("56cbfd0aa3157e000676e1d1"), "item" : "IJK2", "details" : { "model" : "14Q2", "manufacturer" : "M5 Corporation" }, "stock" : [ { "size" : "S", "qty" : 5 }, { "size" : "L", "qty" : 1 } ], "category" : "houseware" }
```

# Query and Projection – Comparison

Name	Description
<code>\$eq</code>	Matches values that are equal to a specified value.
<code>\$gt</code>	Matches values that are greater than a specified value.
<code>\$gte</code>	Matches values that are greater than or equal to a specified value.
<code>\$lt</code>	Matches values that are less than a specified value.
<code>\$lte</code>	Matches values that are less than or equal to a specified value.
<code>\$ne</code>	Matches all values that are not equal to a specified value.
<code>\$in</code>	Matches any of the values specified in an array.
<code>\$nin</code>	Matches none of the values specified in an array.

# Query and Projection – Logical

Name	Description
\$or	Joins query clauses with a logical <b>OR</b> returns all documents that match the conditions of either clause.
\$and	Joins query clauses with a logical <b>AND</b> returns all documents that match the conditions of both clauses.
\$not	Inverts the effect of a query expression and returns documents that do <i>not</i> match the query expression.
\$nor	Joins query clauses with a logical <b>NOR</b> returns all documents that fail to match both clauses.

# Query and Projection

## Element

Name	Description
<code>\$exists</code>	Matches documents that have the specified field.
<code>\$type</code>	Selects documents if a field is of the specified type.

## Evaluation

Name	Description
<code>\$mod</code>	Performs a modulo operation on the value of a field and selects documents with a specified result.
<code>\$regex</code>	Selects documents where values match a specified regular expression.
<code>\$text</code>	Performs text search.
<code>\$where</code>	Matches documents that satisfy a JavaScript expression.

# Querying Embedded Document

```
> db.inventory.find({'details.manufacturer':'ABC Company'})  
{ "_id" : ObjectId("56cbfd0aa3157e000676e1d0"), "item" : "MN02", "details" : { "model" : "14Q3", "manufacturer" : "ABC Company" }, "stock" : [ { "size" : "S", "qty" : 5 }, { "size" : "M", "qty" : 5 }, { "size" : "L", "qty" : 1 } ], "category" : "clothing" }
```

# Querying Embedded Document – Update



- ▶ Use update operators to change field values.

```
db.inventory.update(
  { item: "MNO2" },
  {
    $set: {
      category: "apparel",
      details: { model: "14Q3", manufacturer: "XYZ Company" }
    },
    $currentDate: { lastModified: true }
  }
)
```

# Querying Embedded Document Update

- ▶ The nMatched field specifies the number of existing documents matched for the update, and nModified specifies the number of existing documents modified.

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

# Querying Embedded Document– Update an embedded field

```
> db.inventory.update( { item: "ABC2" }, { $set: { "details.model": "14Q2" } }
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

# upsert Option

- ▶ By default, if no document matches the update query, the update() method does nothing.
- ▶ Specify upsert: true,
  - the update() method either updates matching document or documents, or inserts a new document using the update specification if no matching document exists.

# upsert Option

```
> db.inventory.update(
...   { item: "TBD1" },
...   {
...     item: "TBD1",
...     details: { "model" : "14Q4", "manufacturer" : "ABC Company" },
...     stock: [ { "size" : "S", "qty" : 25 } ],
...     category: "houseware"
...   },
...   { upsert: true }
... )
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : ObjectId("56cc0ad9ba380c097eb98f04")
})
```

# SQL & MongoDB

## SQL Schema Statements

```
CREATE TABLE users (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
```

## MongoDB Schema Statements

Implicitly created on first `insert()` operation. The primary key `_id` is automatically added if `_id` field is not specified.

```
db.users.insert( {
    user_id: "abc123",
    age: 55,
    status: "A"
} )
```

However, you can also explicitly create a collection:

```
db.createCollection("users")
```

# SQL & MongoDB



`ALTER TABLE users`

`ADD join_date DATETIME`

Collections do not describe or enforce the structure of its documents; i.e. there is no structural alteration at the collection level.

However, at the document level, `update()` operations can add fields to existing documents using the `$set` operator.

```
db.users.update(  
    { },  
    { $set: { join_date: new Date() } },  
    { multi: true }  
)
```

# SQL & MongoDB

SQL SELECT Statements

MongoDB find() Statements

---

```
SELECT *          db.users.find()  
FROM users
```

---

```
SELECT *          db.users.find(  
FROM users          { status: "A",  
WHERE status = "A"      age: 50 }  
AND age = 50        )
```

---

```
SELECT *          db.users.find(  
FROM users          { $or: [ { status: "A" } ,  
WHERE status = "A"      { age: 50 } ] }  
OR age = 50        )
```

# SQL & MongoDB

## SQL Update Statements

**UPDATE** users

**SET** status = "C"

**WHERE** age > 25

## MongoDB update() Statements

```
db.users.update(
```

```
  { age: { $gt: 25 } },
```

```
  { $set: { status: "C" } },
```

```
  { multi: true }
```

```
)
```

**UPDATE** users

**SET** age = age + 3

**WHERE** status = "A"

```
db.users.update(
```

```
  { status: "A" } ,
```

```
  { $inc: { age: 3 } },
```

```
  { multi: true }
```

```
)
```

# SQL & MongoDB

**SQL Delete Statements**

**DELETE FROM** users

**WHERE** status = "D"

**MongoDB remove() Statements**

**db.users.remove( { status: "D" } )**

**DELETE FROM** users

**db.users.remove({})**



# SESSION 15

## Index in MongoDB

# CONTENTS

- Essential Concepts behind a Database Index
- Indexing and Ordering in MongoDB
- Creating and Using Indexes in MongoDB



# ESSENTIAL CONCEPTS BEHIND A DATABASE INDEX

- Binary tree
  - At most two child nodes (left and right)
  - Parent and leaf node.
- B-tree
  - Generalization of a binary tree.
  - Allows more than two child nodes for a parent
  - Keeps the data sorted
  - Efficient search and data access.
- B+-tree
  - Special variant of a B-tree.
  - All records are stored in the leaves
  - Leaves are sequentially linked.
  - Used to store a database index.



# INDEX

Share

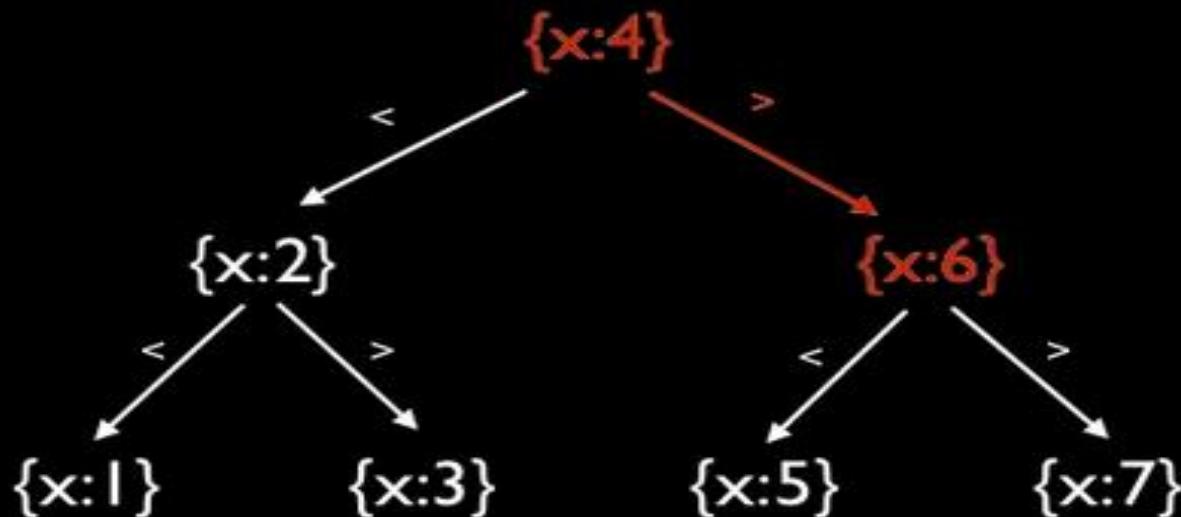
How do we find  $x = 6$  without an index?

{x:0} {x:1} {x:2} {x:3} {x:4} {x:5} {x:6} {x:7} {x:8} {x:9}

6 documents scanned

# INDEX

How do we find  $x = 6$  with an index?



Only 2 documents scanned

# B-TREE STRUCTURE

Index on {a: 1}

$[-\infty, 5)$	$[5, 10)$	$[10, \infty)$
----------------	-----------	----------------

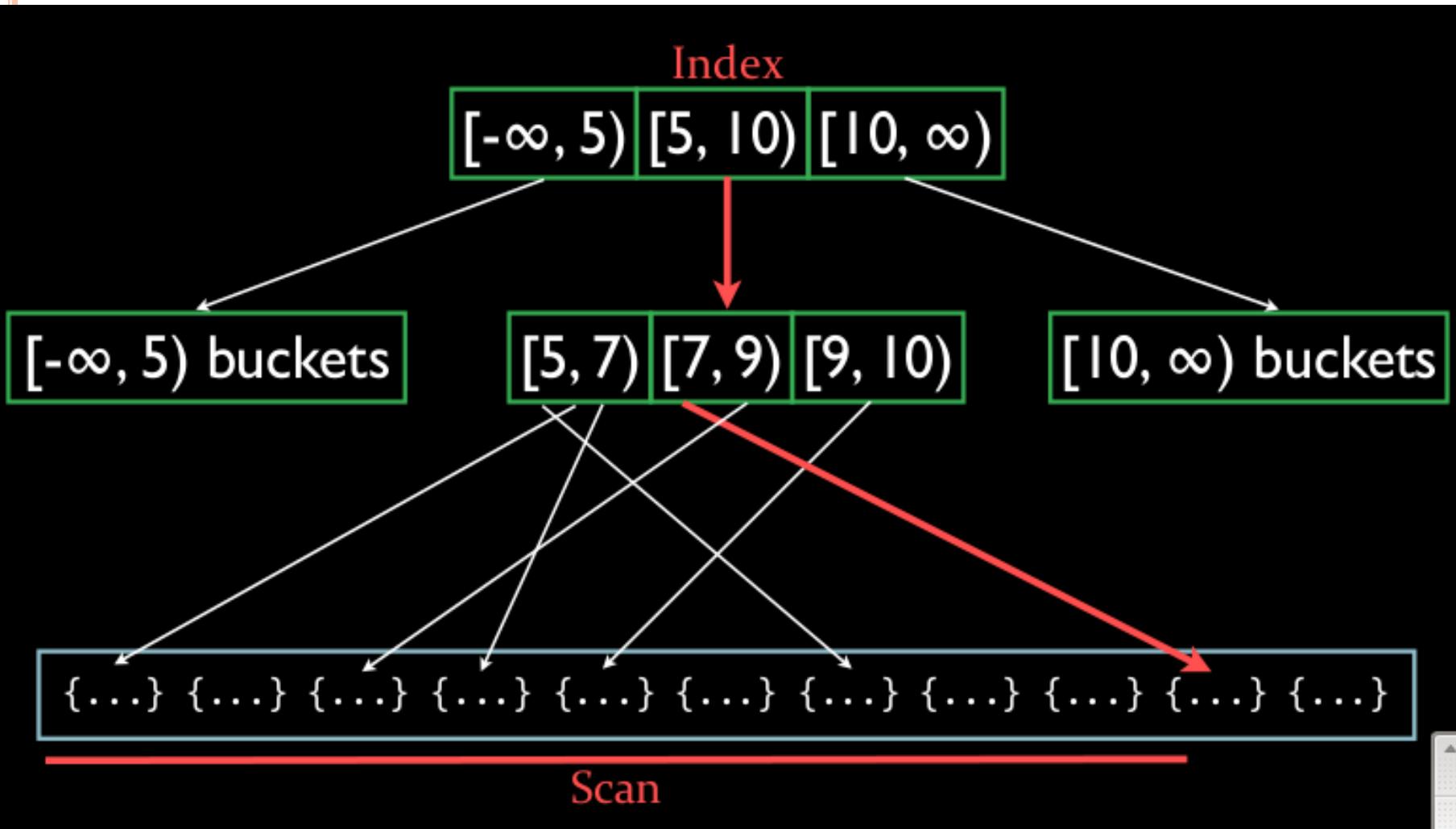
$[-\infty, 5)$  buckets

$[5, 7)$	$[7, 9)$	$[9, 10)$
----------	----------	-----------

$[10, \infty)$  buckets

{...} {...} {...} {...} {...} {...} {...} {...} {...} {...} {...} {...} {...}

# QUERY FOR {A: 7}



# INDEXING AND ORDERING IN MONGODB

- Without indexes, MongoDB must scan every document in a collection to select those documents that match the query statement.
- These collection scans are inefficient
  - Require to process a larger volume of data than an index for each operation.
- Indexes are special data structures
  - Store a small portion of the collections data set in an easy to traverse form.
- Index are used
  - Frequently used queries
  - Low response time
- Drawback
  - Takes Space
  - Slow down writes



# INDEXING AND ORDERING IN MONGODB

- The index stores the value of a specific field or set of fields, ordered by the value of the field.
- MongoDB defines indexes at the collection level and supports indexes on any field or sub-field of the documents in a MongoDB collection.
- MongoDB can use the index to limit the number of documents it must inspect.
- In some cases, MongoDB can use the data from the index to determine which documents match a query.



## WHEN TO INDEX

- Frequently queried field
- Low response time
- Sorting
- Avoid full collection scan



# INDEX TYPES

- Unique Indexes

- Reject duplicate values for the indexed field.
- To create a unique index, use the db.collection.createIndex() method with the unique option set to true.
- The unique constraint applies to separate documents in the collection.
- In the case of a single document with repeating values, the repeated value is inserted into the index only once.
- db.members.createIndex( { "user\_id": 1 }, { unique: true } )



# INDEX TYPES

- TTL Indexes

- use to automatically remove documents from a collection after a certain amount of time
- indexed field must be a date type.
- If the indexed field in a document is not a date or an array that holds a date value(s), the document will not expire.

- db.eventlog.createIndex( { "lastModifiedDate": 1 }, { expireAfterSeconds: 3600 } )



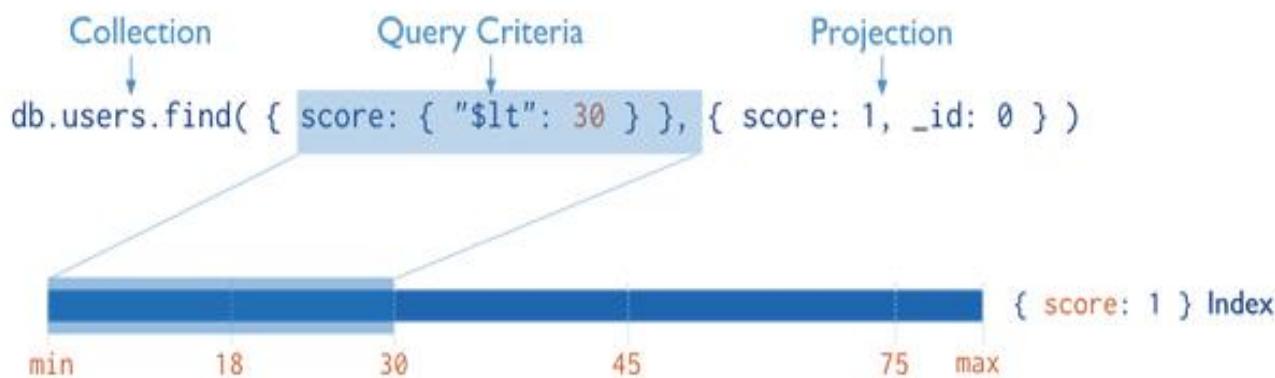
# OPTIMIZATION USING INDEX

- Sorted Results

- documents sorted by the index key directly from the index without requiring an additional sort phase.

- Covered Results

- MongoDB will return results directly from the index without scanning any document



- MongoDB does not need to inspect data outside of the index to fulfill the query

# EXAMPLE

Given the following document in the friends collection:

```
{ "_id" : ObjectId(...),  
  "name" : "Alice"  
  "age" : 27  
}
```

- an index on the name field:
  - db.friends.ensureIndex( { "name" : 1 } )
- Ordering of Indexes
  - ascending, (i.e. 1) or descending (i.e. -1)



# INDEX

- For single-field indexes, the sort order of keys doesn't matter
  - MongoDB can traverse the index in either direction.
- Compound indexes sort order can matter in determining whether the index can support a sort operation.
- Eg;
  - A collection events that contains documents with the fields username and date.
  - Applications can issue queries that return results sorted first by ascending username values and then by descending (i.e. more recent to last)
    - `db.events.find().sort( { username: 1, date: -1 } )`
  - sorted first by descending username values and then by ascending date values, such
    - `db.events.find().sort( { username: -1, date: 1 } )`

# INDEX

- The default name for an index is the concatenation of the indexed keys and each key's direction in the index, 1 or -1
  - create an index on item and quantity:
    - db.products.ensureIndex( { item: 1, quantity: -1 } )
    - The resulting index is named: item\_1\_quantity\_-1.
  - A name can be specified for an index instead of using the default name.
    - Flowing command create an index on item and quantity and specify inventory as the indexname:
      - db.products.ensureIndex( { item: 1, quantity: -1 } , { name: "inventory" } )



# INDEXING AND ORDERING IN MONGODB - OPTIMIZATION

- Create an Index on a Single Field
- The ensureIndex() method only creates an index if an index of the same specification does not already exist.
- For example, the following operation creates an index on the userid field of the records collection:
  - db.records.ensureIndex( { userid: 1 } )
- The created index will support queries that select on the field userid, such as the following:
  - db.records.find( { userid: 2 } )
  - db.records.find( { userid: { \$gt: 10 } } )



# ORDERS ARE MAINTAINED IN INDEX

Index on {a: 1}

```
{a: 0, b: 9}  
{a: 2, b: 0}  
{a: 3, b: 2}  
{a: 3, b: 7}  
{a: 3, b: 5}  
{a: 7, b: 1}  
{a: 9, b: 1}
```

Index on {a: 1, b: -1}

```
{a: 0, b: 9}  
{a: 2, b: 0}  
{a: 3, b: 7}  
{a: 3, b: 5}  
{a: 3, b: 2}  
{a: 7, b: 1}  
{a: 9, b: 1}
```

# INDEX TYPES

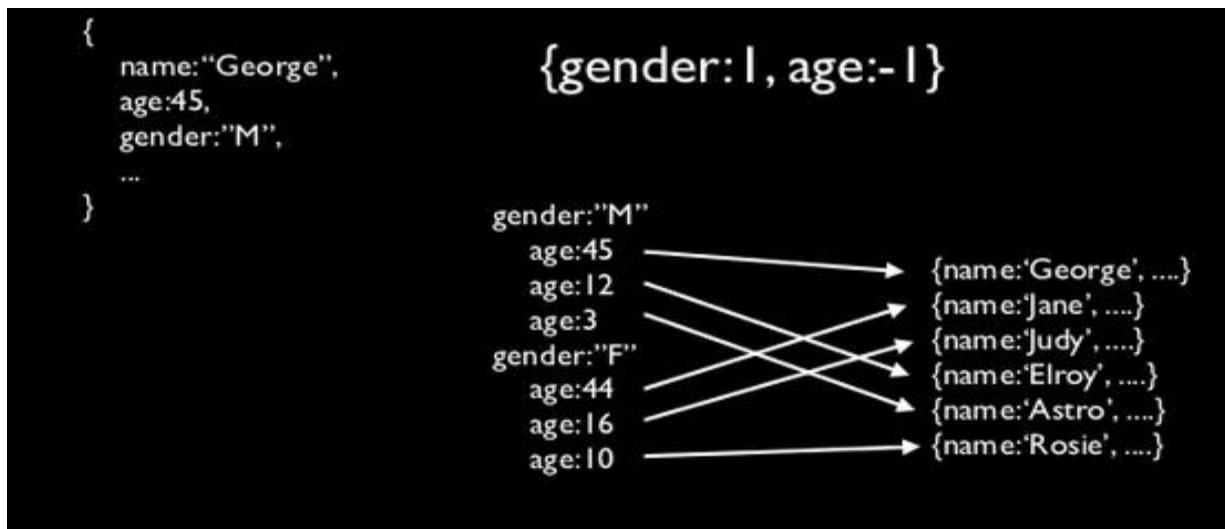
- Default `_id`
  - All MongoDB collections have an index on the `_id` field that exists by default.
  - If applications do not specify a value for `_id` the driver or the mongod will create an `_id` field with an ObjectId value.
  - The `_id` index is unique, and prevents clients from inserting two documents with the same value for the `_id` field.
- Single Field
  - MongoDB supports user-defined indexes on a single field of a document.



# COMPOUND INDEX

For example

- an application that stores data about customers.
- To find customers based on last name, first name, and state of residence.
- With a compound index on last name, first name and state of residence, queries could efficiently locate people with all three of these values specified.



# COMPOUND INDEX

- User-defined indexes on multiple fields.
- The query can select documents based on additional fields.
- The order of fields listed in a compound index has significance.
  - { userid: 1, score: -1 }
  - the index sorts first by userid and then, within each userid value, sort by score.
- To create a compound index use an operation that resembles the following prototype:
  - db.collection.ensureIndex( { a: 1, b: 1, c: 1 } )
- Example
  - The following operation will create an index on the item, category, and price fields of the products collection:
  - db.products.ensureIndex( { item: 1, category: 1, price: 1 } )

# INDEX OPERATIONS

- Listing indexes
  - db.user.getIndexes();
- Dropping indexes
  - db.user.dropIndex()
- Background Building Indexes
  - db.user.ensureIndex( { ... }, {background : true} )



# MULTIKEY INDEX

- MongoDB uses multikey indexes to index the content stored in arrays.
- If you index a field that holds an array value, MongoDB creates separate index entries for every element of the array.
- These multikey indexes allow queries to select documents that contain arrays by matching on element or elements of the arrays.
- MongoDB automatically determines whether to create a multikey index if the indexed field contains an array value



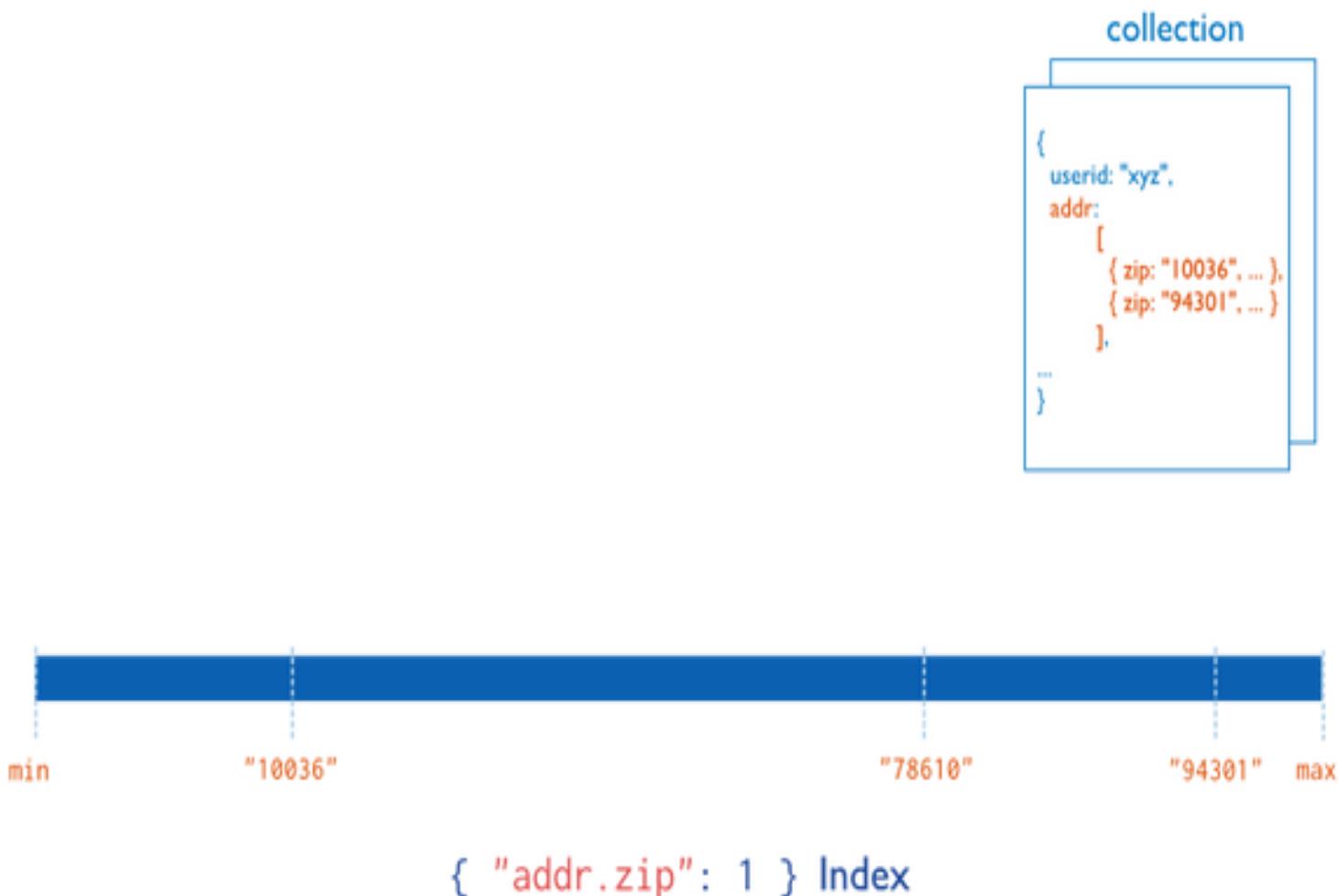


Diagram of a multikey index on the `addr.zip` field. The `addr` field contains an array of address documents. The address documents contain the `zip` field.

# DROP DUPLICATES

- MongoDB cannot create a unique index on a field that has duplicate values.
- To force the creation of a unique index
  - dropDups option, which will only index the first occurrence of a value for the key, and delete all subsequent values.
- To create a unique index that drops duplicates on the username field of the accounts collection
  - db.accounts.ensureIndex( { username: 1 }, { unique: true, dropDups: true } )



# CREATE INDEXES TO SUPPORT YOUR QUERIES

- An index supports a query when the index contains all the fields scanned by the query.
- The query scans the index and not the collection.
- Creating indexes that support queries results in greatly increased query performance.
- Create a Single-Key Index if All Queries Use the Same, Single Key
  - If you only ever query on a single key in a given collection, then you need to create just one single-key index for that collection.
  - For example, you might create an index on category in the product collection:
    - db.products.createIndex( { "category": 1 } )



# CREATE INDEXES TO SUPPORT YOUR QUERIES

- If query is on only one key and at other times query on that key combined with a second key, then creating a compound index is more efficient than creating a single-key index.
- MongoDB will use the compound index for both queries.
  - For example, you might create an index on both category and item.
  - `db.products.createIndex( { "category": 1, "item": 1 } )`



# REMOVE INDEX

- db.collection.dropIndex() method.
  - To remove an index

Example,

- db.accounts.dropIndex( { "tax-id": 1 } )
- removes an ascending index on the tax-id field in the accounts collection:
- The operation returns a document with the status of the operation:
- { "nIndexesWas" : 3, "ok" : 1 }
- Where the value of nIndexesWas reflects the number of indexes before removing this index.



# EXPLAIN

- Use the db.collection.explain()

- To return statistics about the query process, including the index used, the number of documents scanned, and the time the query takes to process in milliseconds.
- To provide information on the execution of other operations, such as db.collection.update(). See db.collection.explain() for details.



# LIMITATIONS

- Every write (insert, update or delete) will take longer for every added index
  - MongoDB has to update all the index whenever data changes
- Index requires two lookups
  - One to look at the index entry and one following index's pointer to document
- Use Index for
  - Large collections
  - Large documents
  - Selective queries



# SUMMARY

Name	Description
<code>db.collection.createIndex()</code>	Builds an index on a collection.
<code>db.collection.dropIndex()</code>	Removes a specified index on a collection.
<code>db.collection.dropIndexes()</code>	Removes all indexes on a collection.
<code>db.collection.getIndexes()</code>	Returns an array of documents that describe the existing indexes on a collection.
<code>db.collection.reIndex()</code>	Rebuilds all existing indexes on a collection.
<code>db.collection.totalIndexSize()</code>	Reports the total size used by the indexes on a collection. Provides a wrapper around the <code>totalIndexSize</code> field of the <code>collStats</code> output.
<code>cursor.explain()</code>	Reports on the query execution plan for a cursor.
<code>cursor_hint()</code>	Forces MongoDB to use a specific index for a query.
<code>cursor.max()</code>	Specifies an exclusive upper index bound for a cursor. For use with <code>cursor_hint()</code>
<code>cursor.min()</code>	Specifies an inclusive lower index bound for a cursor. For use with <code>cursor_hint()</code>