Course Notes

Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

COMP5531

Class Notes by Bipin. C. Desai

Pl. see: https://users.encs.concordia.ca/~bcdesai/CopyForward.pdf

Co:ordinates

Dept. of Comp. Sci & Soft. Engg.

Concordia University, Montreal

Any email message sent to me, from a ‘free’ email service or a ‘smart’ device, will be deleted by my email filter and hence would never be read much less replied..

Facebook Expects to Be Fined Up to $5 Billion by F.T.C.

https://www.nytimes.com/2019/04/24/technology/facebook-ftc-fine-privacy.html

Facebook fined for data breaches in Cambridge Analytica

www.theguardian.com/technology/2018/jul/11/facebook-fined-for-data-breaches-in-cambridge-analytica-scandal

Google hit with £44m GDPR fine over ads

https://www.bbc.com/news/technology-46944696

Google fined record €2.4bn by EU over search engine results

https://www.theguardian.com/business/2017/jun/27/google-braces-for-record-breaking-1bn-fine-from-eu

Google Fined Record $5 Billion by EU, 90 Days to Stop 'Illegal Practices'

https://www.bloomberg.com/news/articles/2018-07-17/
google-said-to-have-11th-hour-call-with-eu-ahead-of-android-fine

Google va payer plus de 150 millions de dollars - données des enfants

https://ici.radio-canada.ca/nouvelle/1282501/
Google-youtube-protection-donnees-enfants

High Tech Extortion Racket

Harpers Magazine, September 2020

Meta offers Canadian Facebook users $51M to settle lawsuit in 4 provinces

Legal action started in **2012** - Debbie Douez sues Facebook for using her photo in an advertisement.

Twitter Sues Nonprofit That Tracks Hate Speech

Center for Countering Digital Hate said it had received a letter accusing it of trying to hurt the social platform with its research

OSN and

Group work

Form group: of 4

Assignment 0: Must be done before end of week 1 : Max marks 0.5

Form a group and register in CrsMgr:

Change email address to ENCS account

Choose a leader or vote for one.

Choice (1): Form your group by uploading a file that contains the student information of your group members. Choose a group leader

If you intend to lock you group right after your group is created, this is the right choice. Other students are unable to join your group unless it is unlocked by one of the group members.

Choice (2): Join one of the existing course groups that are not full and not locked!

Vote for group leader

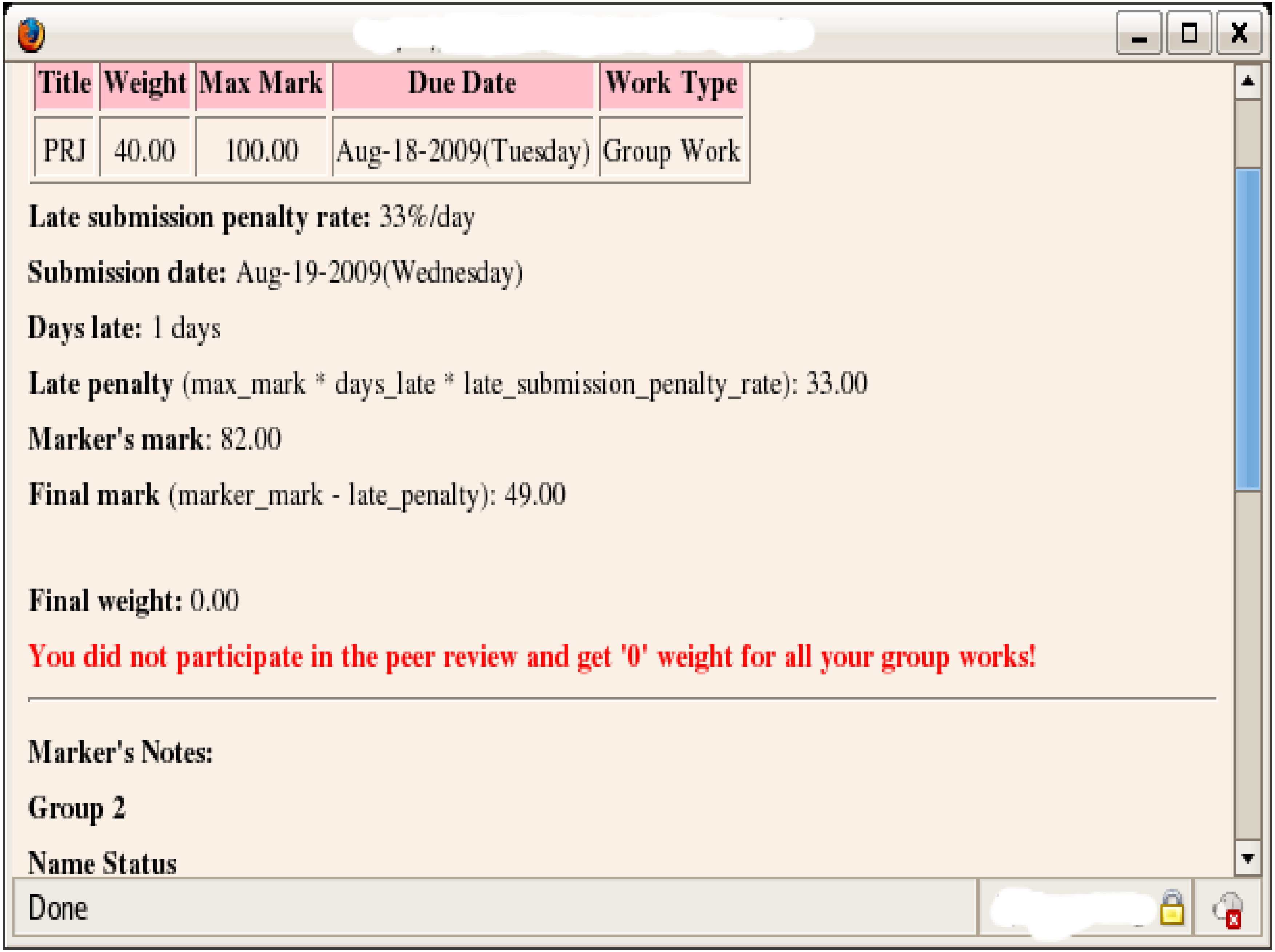
FORMING A GROUP (OFF-LINE) AND UPLOADING THE GROUP INFORMATION

Write the student information of your members in a text file according to the following format: (the information for the group leader must be put in the first line):

Student id of the Group Leader
Student id of 2nd Group Member
...
Student id of the last Group Member

When the file is uploaded, you would be shown your group and are required to confirm it.

Example of what happens if YOU do not enter a peer evaluation



Title	Weight	Max Mark	Due Date	Work Type
PRJ	40.00	100.00	Aug-18-2009(Tuesday)	Group Work

Late submission penalty rate: 33%/day

Submission date: Aug-19-2009(Wednesday)

Days late: 1 days

Late penalty (max_mark * days_late * late_submission_penalty_rate): 33.00

Marker's mark: 82.00

Final mark (marker_mark - late_penalty): 49.00

Final weight: 0.00

You did not participate in the peer review and get '0' weight for all your group works!

Marker's Notes:

Group 2

Name Status

Done

Bipin C Desai

What happens if YOU enter an “illegal” peer evaluation

The range of peer evaluation is between 0 1nd 100 and can be done for a marked entity as soon as it is posted

Deadline for the peer evaluation is within one day of the marked entity’s due date

The range of peer evaluation is between 0 to 100

Any value outside this ‘honorable’ range would is a violation of this rule

ALL peer evaluations, for the marked entity, submitted outside the legal range are deleted and the person making even a single such non-honorable mark is considered NOT to have done that peer evaluation

Objectives

- Concepts and use data models(E-R, RDM, unstructured)
- Intro. To the Relational database management systems (RDBMS)
- Query languages (Relational Algebra & Calculus, SQL)
- Concepts of checks, assertions, and triggers.
- Database design and web-programming including: HTML, Javascript, PhP – design/implementation of a real application.

The extent to which you would master these depends entirely on **you**:
A conductor does not play any instrument, nor does the coach play on the ice/field.

A medal (any colour) needs lots of training –before the event!

Counting & calculating

Data base

Data ba$$$$$e

https://commons.wikimedia.org/w/index.php?curid=4954357

Data

Where does data come from?

Records of status
Operation of organizations
Data + action → more data

What to record
- Legal and traditional commitments
- cost of recording and preserving the records

Data + Algorithm → Programs +++

Computers

Jacquard machine - 1804
Textile weaving machine
  Pattern controlled by punched cards (first input device)

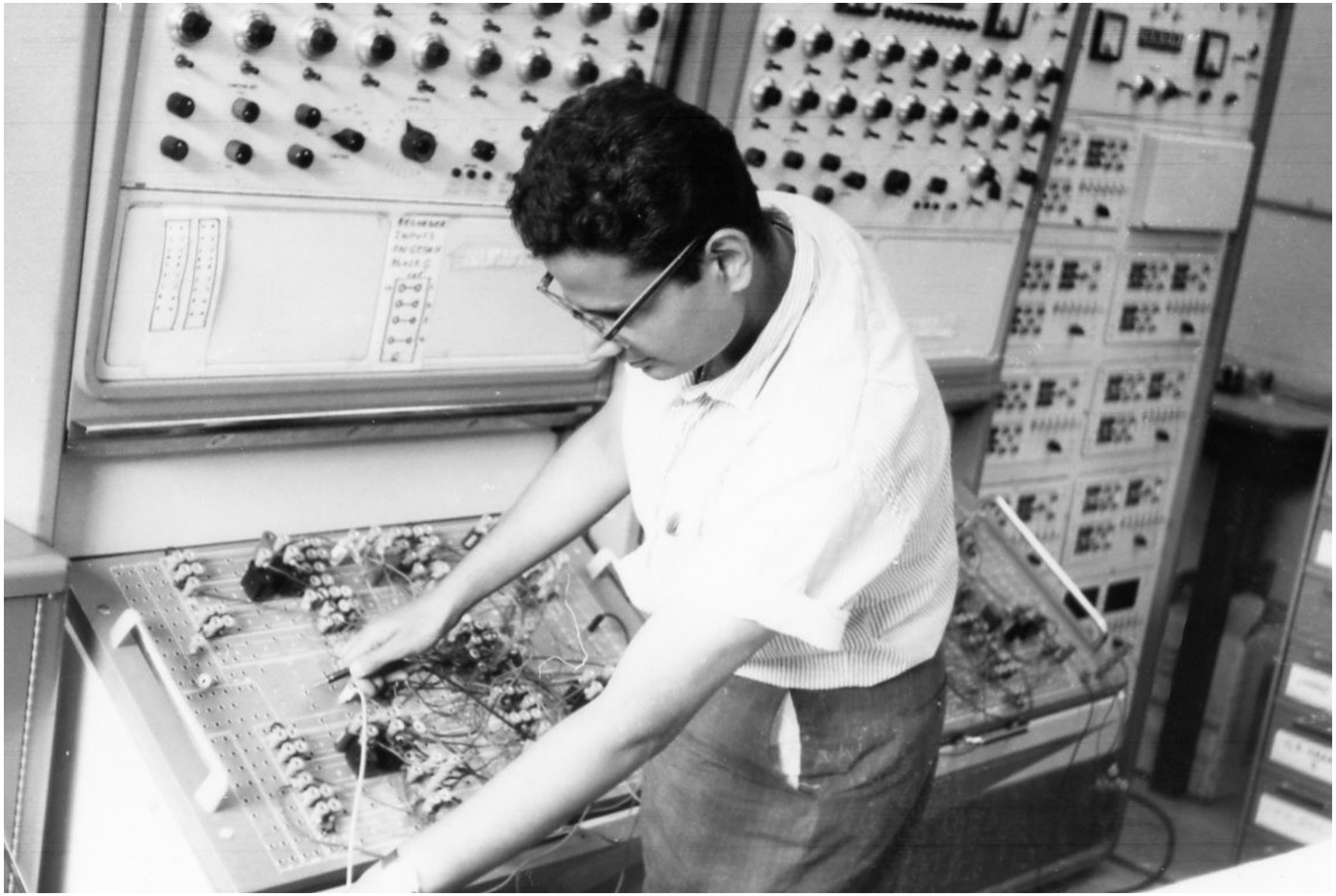
Charles Babbage:
Difference Engine – 1832
Analytical Engine - no funds
    Processor, storage, input device, output device
    Partially completed in 1910
    Fully programmable
First programmer: Lady Lovelace -Ada (daughter of Byron- the poet)

US Census-
 Hollerith and the tabulating machine – used punched cards
1911 -Computing-Tabulating-Recording Company ► IBM

Computers

1930 Vennevar Bush ► an analog computer ►Differential Analyzer

Programming an analog computer



Computers

1934 James Hilton wrote GoodBye Mr. Chips in 4 days-£50 a royalty!

1939-1843 Howard Aiken and IBM

- ► Mark I (mechanical) Mark IV (vacuum tube)

1939 Atanasoff/Berry Iowa State

1943 Digital computer- Colossus (UK)

1945 Eniac (USA)

1949 Manchester Mark I

https://www.britannica.com/technology/computer/The-first-computer

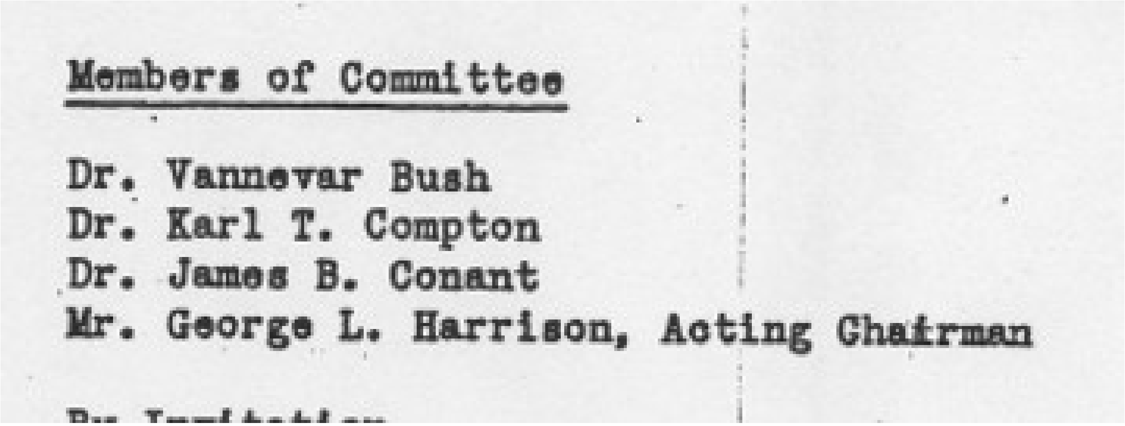
1945

Feb. 4-11 Yalta conference: Decide Europe’s re-organization

April 30 Hitler commits suicide

May 8 Armistice day

March-July: Interim committee for the deployment of the Atomic bomb,



Members of Committee

Dr. Vannevar Bush
Dr. Karl T. Compton
Dr. James B. Conant
Mr. George L. Harrison, Acting Chairman

By Invitation

July: Final report of the interim cmt. & a press statement on dropping the bomb(s)

July: Publication(The Atlantic, July, 1945) of *As We May Think* by VANNEVAR BUSH concept of linked documents

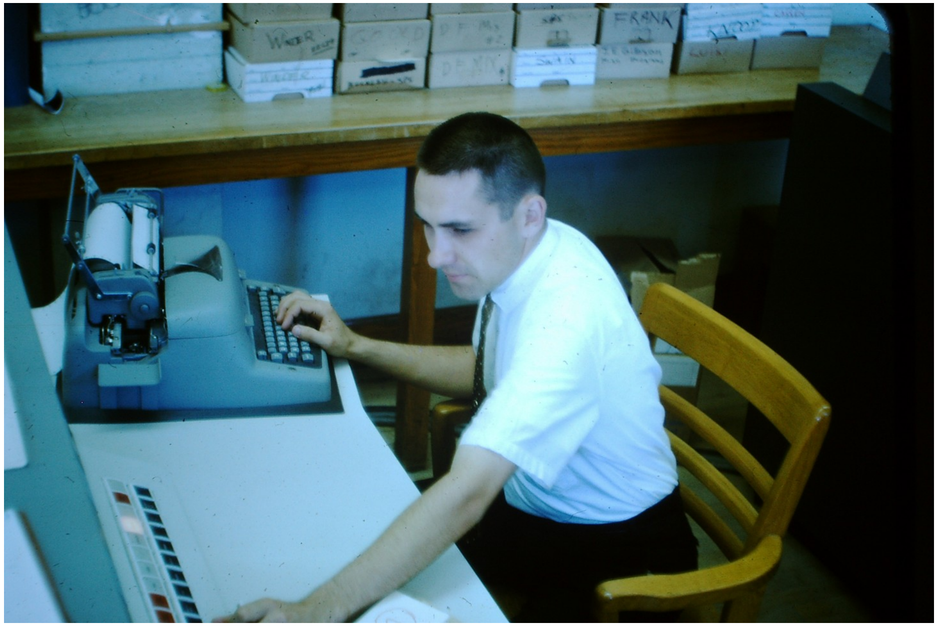
https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/

August: Most of the Japanese forces have been defeated

August 6 US drops atomic bomb on Hiroshima

August 9 US drops atomic bomb on Nagasaki

Late 1940s
End of colonization, and new ones?
https://en.wikipedia.org/wiki/Decolonisation_of_Asia - /Nakba
1950s
Development of the digital computers
1960
Concept of wide area network, ARPANET, packet transmission
Early databases
1970s
TCP/IP protocol, Relational Databases, SQL – IBM a late R-DBMS starter
Time sharing, multi-tasking. IBM and the effective end of Usain anti-monopoly law applications. Birth of PC and DOS -drop-out kid wonder!
1980s
Commercialization of the internet, ISPs,
Hypertext, HTML, HTTP, OODB
1990s
Web browsers, search engines, massive data collection, tracking
21st Century
NoSQL, OSN, more drop-out billionaires, AI, LLM, Internet colonization

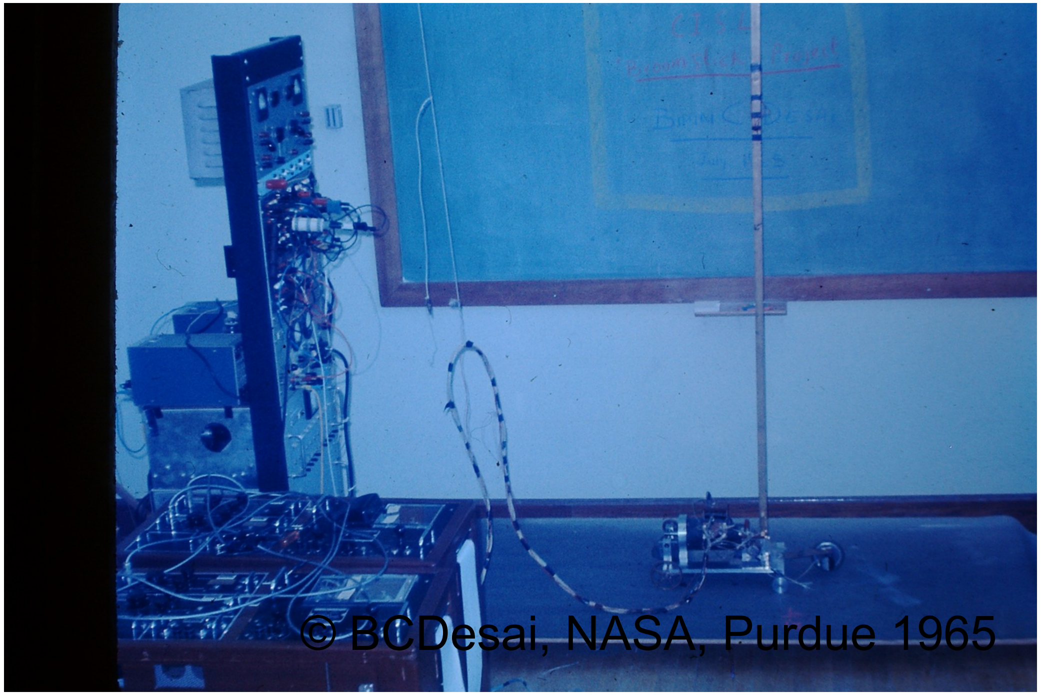


Allan Gondeck on an IBM1620 1964

https://en.wikipedia.org/wiki/IBM_729

IBM 026 keypunches, IBM 1403 line printer and IBM 729 tape drives
https://commons.wikimedia.org/w/index.php?curid=17267381

IBM 7090 computer
https://commons.wikimedia.org/w/index.php?curid=2878809



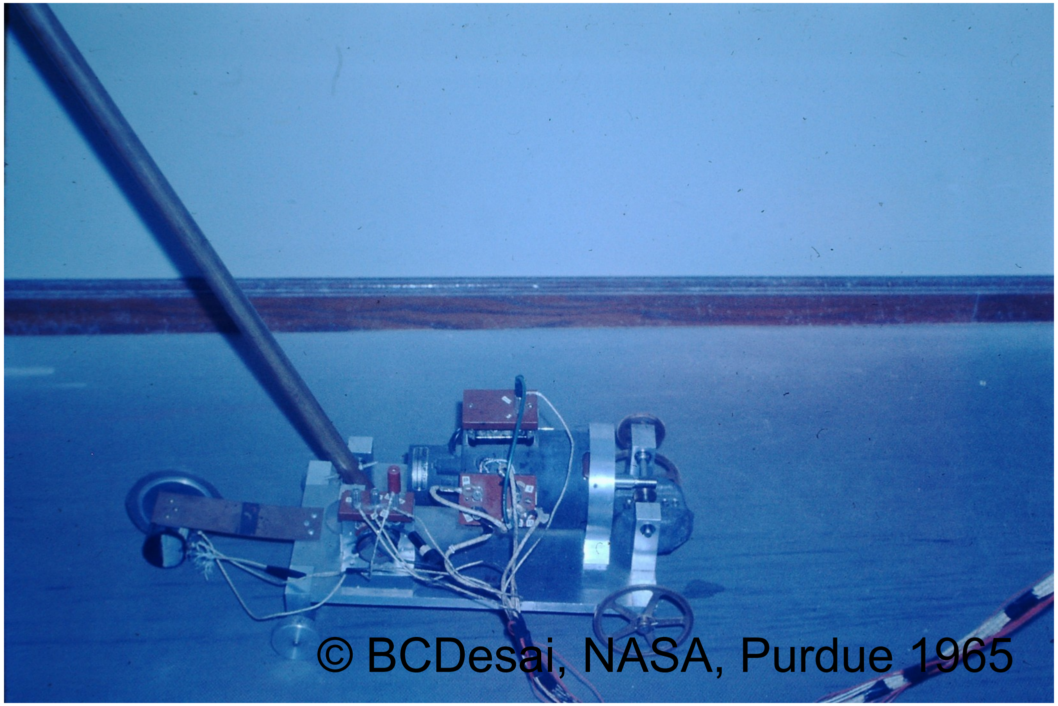
© BCDesai, NASA, Purdue 1965

Modelling:

Represent (approximate)
-physical thing,
-conceptual thing

Broomstick stability control

Application:
Stability of rockets such as: Saturn V



© BCDesai, NASA, Purdue 1965

Saturn V - The Real thing – 1969

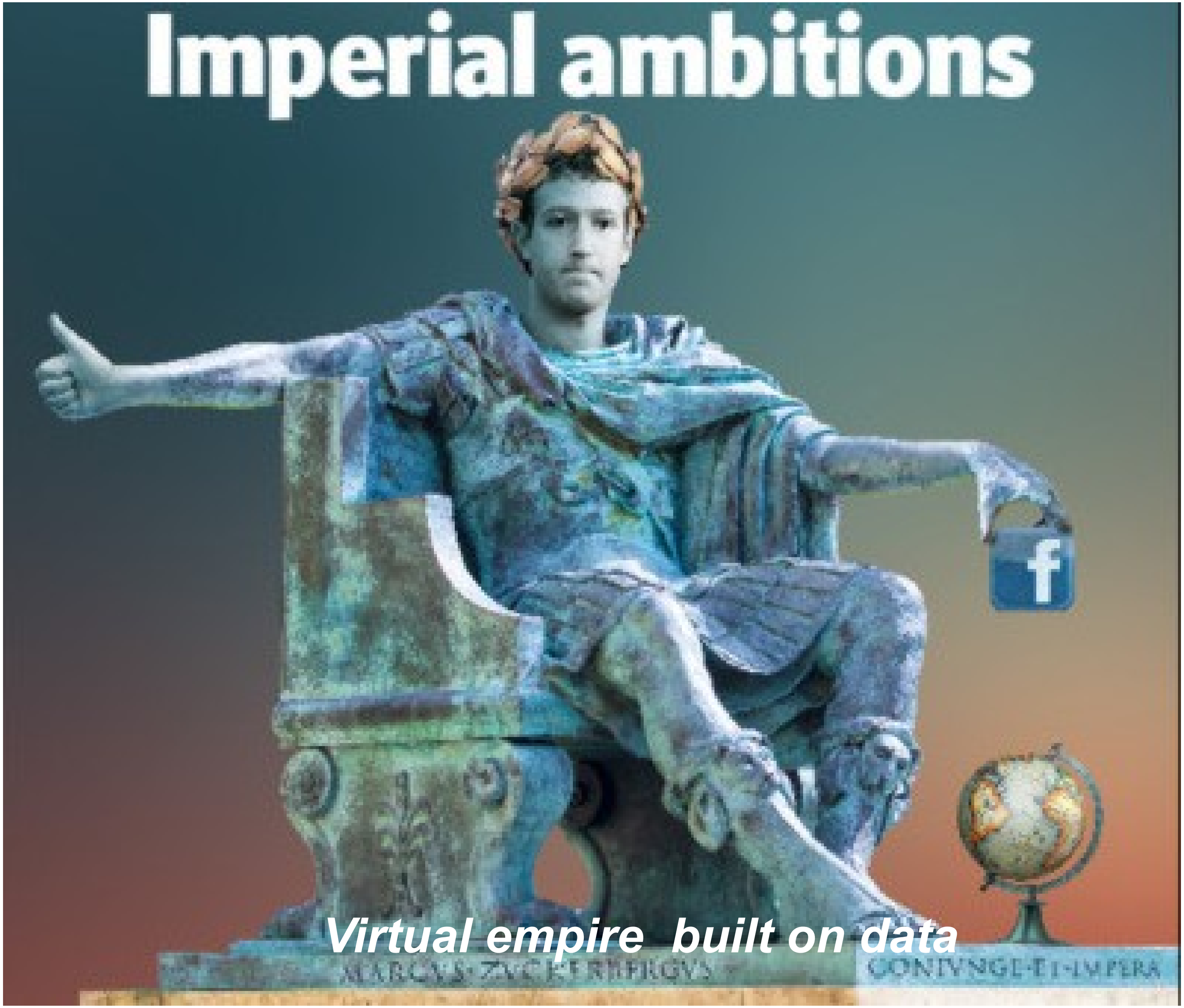
wikimedia.org/w/index.php?curid=6448924

Data deluge and exploitation:
The Beginning!

Børre Ludvigsen, Dr. Bipin Desai; Dr. Yuri Rubinsky

http://www94.web.cern.ch/WWW94/Images/ClosingPanel/Closingpanel1.html

Constantine the Great
https://www.flickr.com/photos/yorkminster/5390106900/



Virtual empire built on data

https://www.economist.com/leaders/2016/04/09/imperial-ambitions

Course Notes

Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

This document in electronic form, bearing a CopyForward permission, could be used for personal use and/or study, free of charge. Anyone could use it to derive updated versions. The derived version must be published under CopyForward. All authors of the version used to derive the new version must be included in the updated version in the existing order, followed by name(s) of author(s) producing the derived work

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

Introduction: DBMS

Bipin C. DESAI

Pl. see: https://users.encs.concordia.ca/~bcdesai/CopyForward.pdf

Grading:

As per the administrative slides

Only random parts (which could be all or some or none) of the assignments could be graded!

The final letter grade would be based on the traditional conversion scheme; e.g, A's would be assigned to numerical total marks well above high 80s

Office hours: As in CrsMgr
For common questions answers would be posted on the announcements/FAQ entries for the course page in CrsMgr page. Pl. read/re-read these before sending an email. Pl. send email in **text**
DO NOT SEND DUPLICATE emails

Tutors, Lab Instructors : As announced on CrsMgr

Account for course manager has been created and the ID/PW have been emailed!
If you have not seen it, look in your spam/thrash folder.
Late registration will have to wait for their records to be exported to CrsMgr.
For the students who haven't provided an email to the Concordia's SIS, you need to find out your ID/PW!
Sign in to CrsMgr: change your PW & update email address

Course Manager System(CrsMgr):
https://confsys.encs.concordia.ca/CrsMgr

If you are officially registered in the course, you would be registered in CrsMgr.
CrsMgr would be used for administrating on-line quizzes, managing the grades, submission of assignments and reports, scheduling of demos and peer evaluation for each group marked entity.

Your grade and the feedback could be viewed in CrsMgr.
Please read the announcements for this course regularly in CrsMgr.
Answers to common questions as well additional instructions
for the course would be posted in the FAQ section.
Look at it before sending out emails.
No emails will be answered if the instructions are already posted.
Pl. send email in text – DOES NOT MEAN TEXT MESSAGES

Sign in to CrsMgr and change your email adr. to the ENCS email.

Form a team of 4 in the first class or before the deadline:

Choose a leader: the leader signs into CrsMgr and joins a new group

- update his/her email to **ENCS** computer account
- ~~insert a password for group DB~~ (would be generated automatically):

Each member of the team then joins this group.

For each member of the team:

- update his/her email to **ENCS** computer account

Above steps could be done by using and uploading a text file!

If you do not join a group, youwill be put randomly in any group to create groups with up to 4 members

Note: Assignments and Projects (warm up and main) would be done by this group with a single submission per group to be uploaded by the group(team) leader.

Upload the assignment/project to the course manager system

https://confsys.encs.concordia.ca/CrsMgr

ONLY the group leader could upload a group submission.

Software

On the ENCS system:

MySQL/MariaDB (Oracle is no longer supported by AITS!)
PHP
HTML
CGI (security) – defunct: each group have their own system

On you own(for WinX or Linux)

Download and install Mariadb(opensource version of MySQL)
https://mariadb.org/
Install PHP: http://www.php.net/
Install Web Server: http://www.apache.org/

Remember the demo would be on the ENCS system so you need to port your code/database Your system must also work on a Linux platform.
AITS uses MySQLm

Look into LAMP or its port to Windows; the following may have changed!

https://ampps.com/download
https://codebriefly.com/how-to-setup-apache-php-mysql-on-windows-10/

If you use other database engines:

NO HELP from us

Only MySQL is supported by ENCS's AITS

You are free to use any database engine.

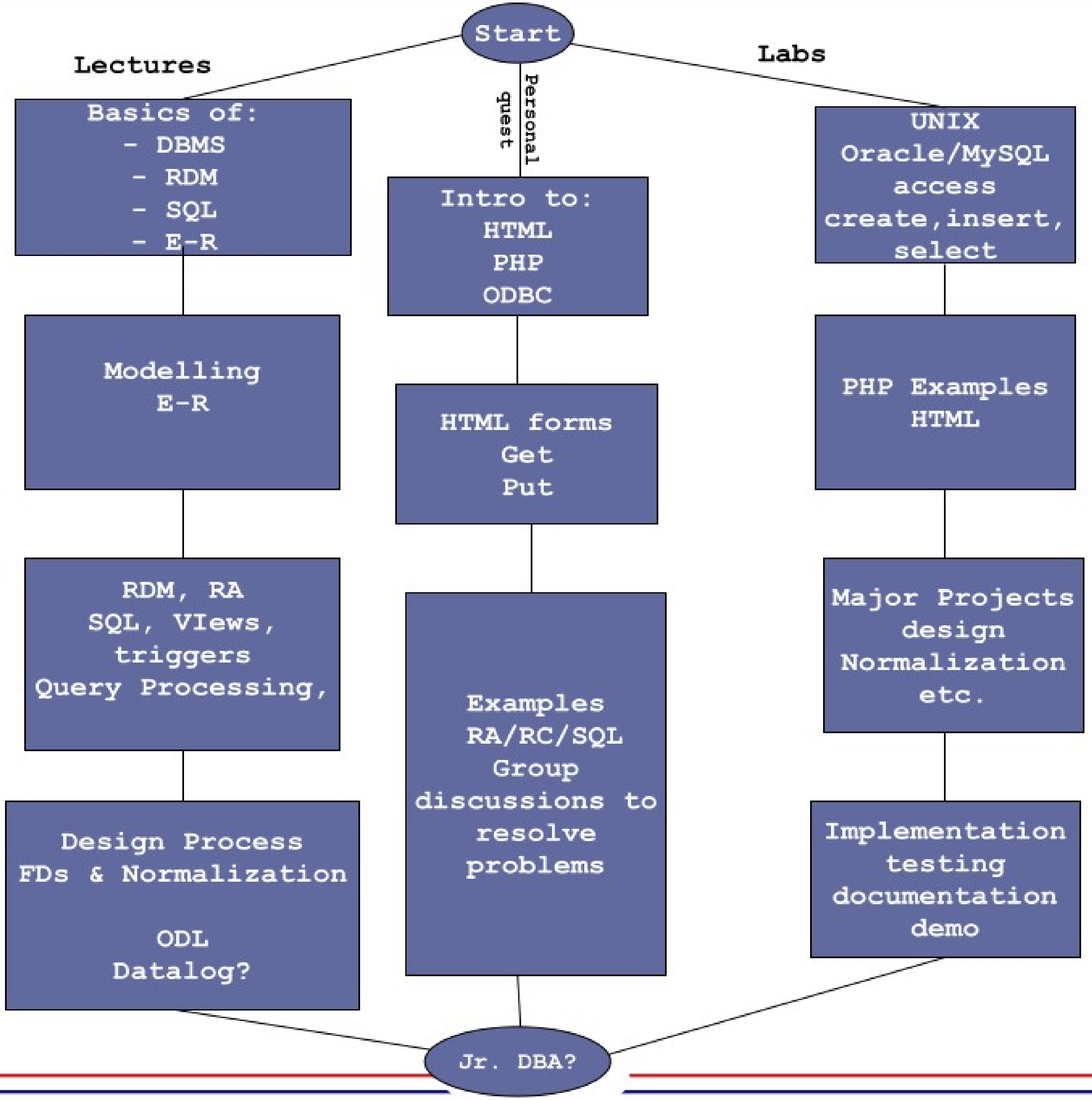
However, you are on your own to download and install the database on your own system;

You also need to make arrangements with your teammates so that the work is coordinated.

You need to install PhP and Apache servers as well.

The database system applications you develop for the projects **must** be compatible with Linux (optionally WinX) and ported to one of our system for the demo.

Remember the demo would be on the ENCS system so you need to port your projects!



Start

Lectures

Basics of:
- DBMS
- RDM
- SQL
- E-R

Modelling
E-R

RDM, RA
SQL, VIews,
triggers
Query Processing,

Design Process
FDs & Normalization

ODL
Datalog?

Personal quest

Intro to:
HTML
PHP
ODBC

HTML forms
Get
Put

Examples
RA/RC/SQL
Group discussions to resolve problems

Labs

UNIX
Oracle/MySQL
access
create,insert,
select

PHP Examples
HTML

Major Projects
design
Normalization
etc.

Implementation
testing
documentation
demo

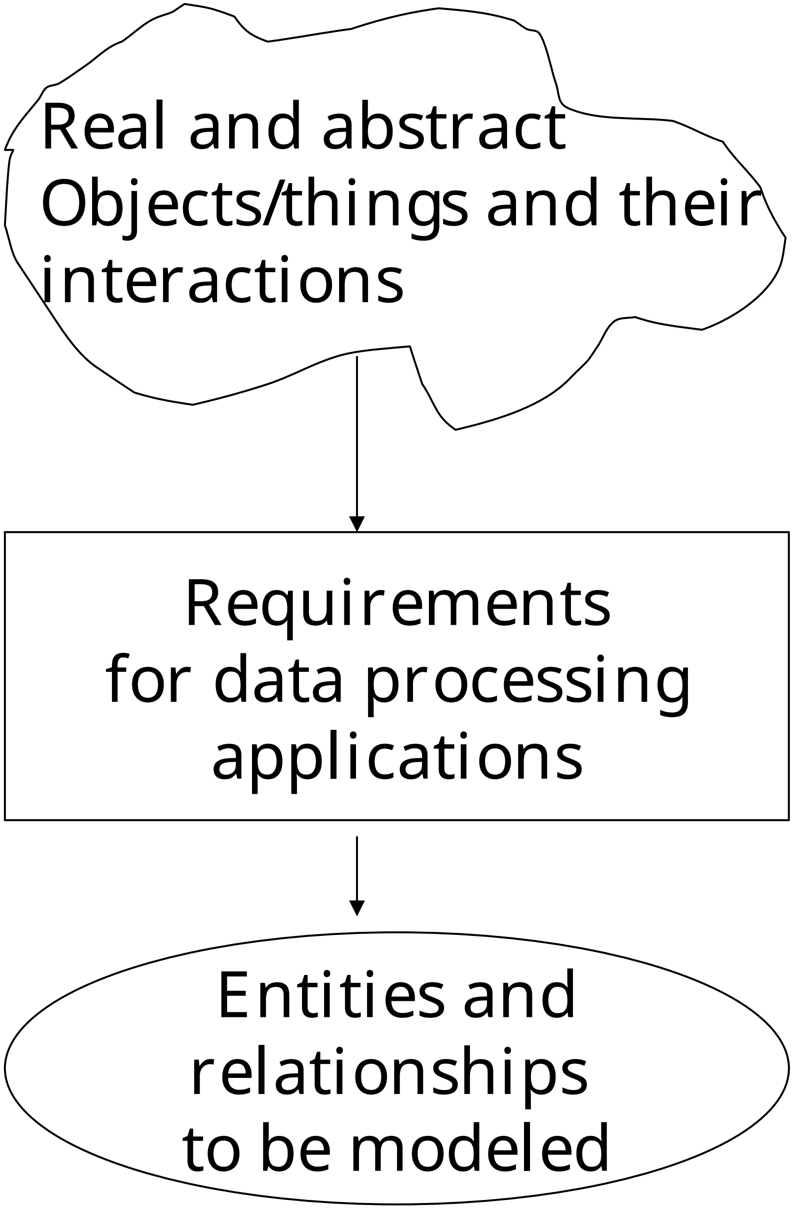
Jr. DBA?

Modeling techniques (E-R, ODL)

Basic relational model

Design of database applications

Database programming (MySQL, SQL, PHP, HTML, CSS, Javascript)



Real and abstract Objects/things and their interactions

Requirements for data processing applications

Entities and relationships to be modeled

DBMS! What is it?

Database is an integrated data collection (Logically consistent and persistent)

It is derived from the model of a set of applications for a real world enterprise.

DBMS is a software package designed to make managing almost any database.

DBMS offers: data independence, efficiency, integrity, security, concurrency, recovery

Why Database?

Information Age: 30-40% of world trade and growing

Web(Unstructured data) and .com

Digital Library

Human Genome Project

Email,
Entertainment
OSN,
Shopping

Day to day operation of Mama/Papa Store

List of titles, artist, and download site of shared files.

Information about employees, departments, projects, etc. in an organization

Information about students, courses, enrollments, professors, etc. in an educational institute

Information about books, videos, albums, members, etc. in a library

DBMS is a complex set of software packages:

- create new databases, store and manage data
- provide application development and support environment

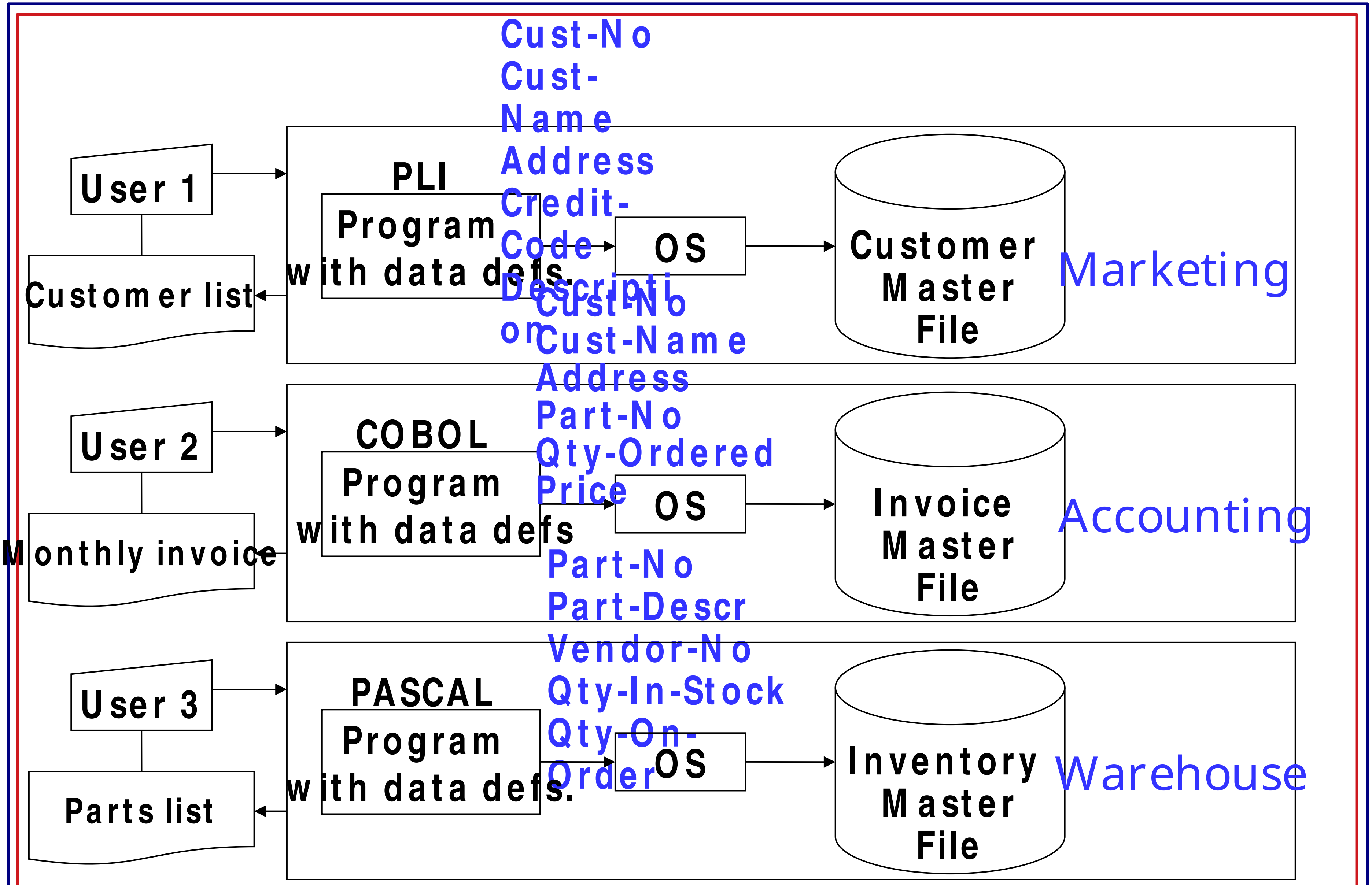
Application Support: Gives developers tools to build applications for using the data. Allows easy method for users to query and modify the data

Persistent storage: Support the storage of data

Transaction management: Controls concurrent access to data from many users

Supports the ACID properties .

Atomicity Concurrency Integrity Durability


User 1
Customer list
PLI
Program
with data defs.
Cust-No
Cust-Name
Address
Credit-Code
Description
OS
Customer Master File
Marketing
User 2
Monthly invoice
COBOL
Program
with data defs
Cust-No
Cust-Name
Address
Part-No
Qty-Ordered
Price
OS
Invoice Master File
Accounting
User 3
Parts list
PASCAL
Program
with data defs.
Part-No
Part-Descr
Vendor-No
Qty-In-Stock
Qty-On-Order
OS
Inventory Master File
Warehouse

Pros & Cons of file based system

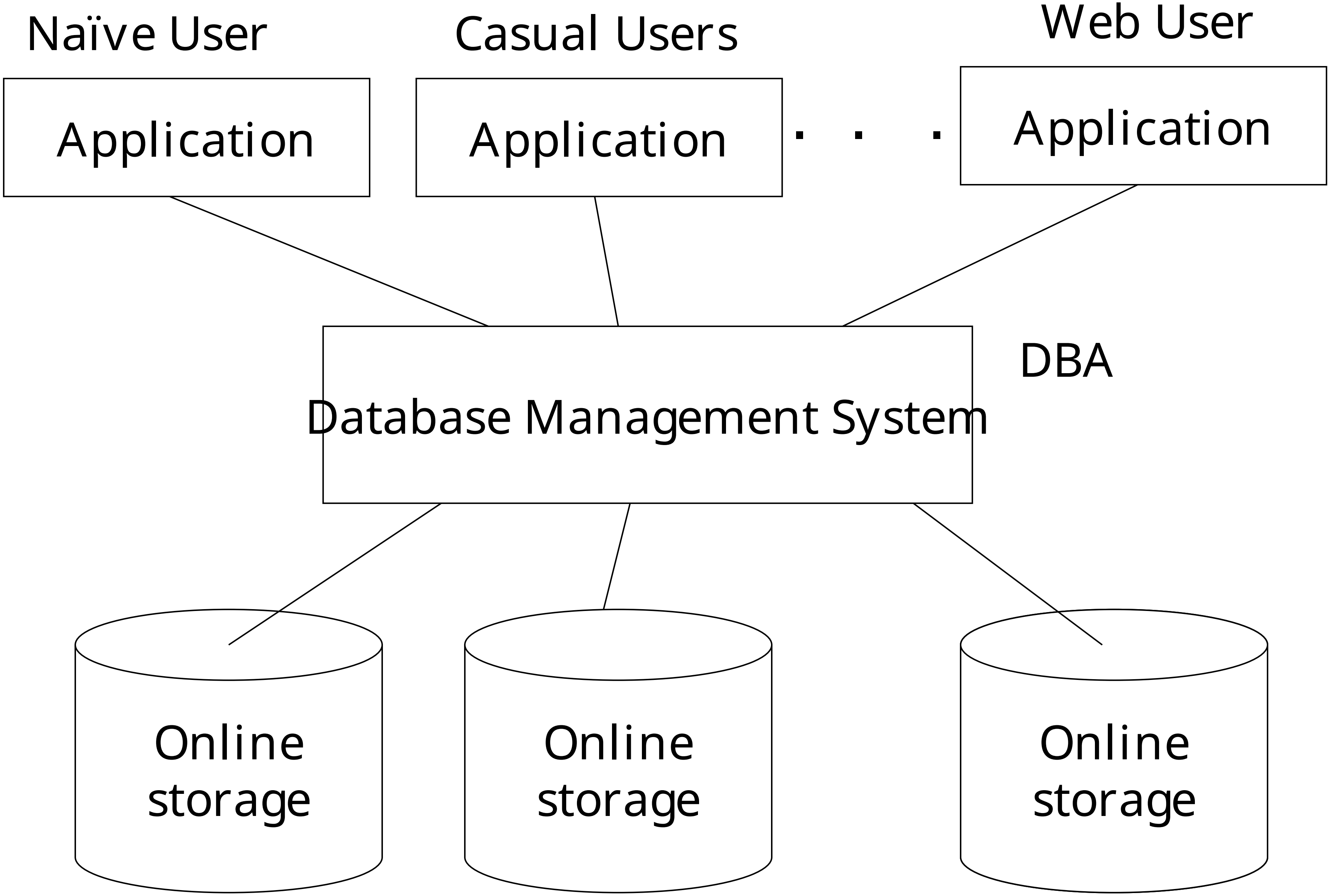
Sharing not possible data definition is “locked” in application programs which “owns” the file and the data in it

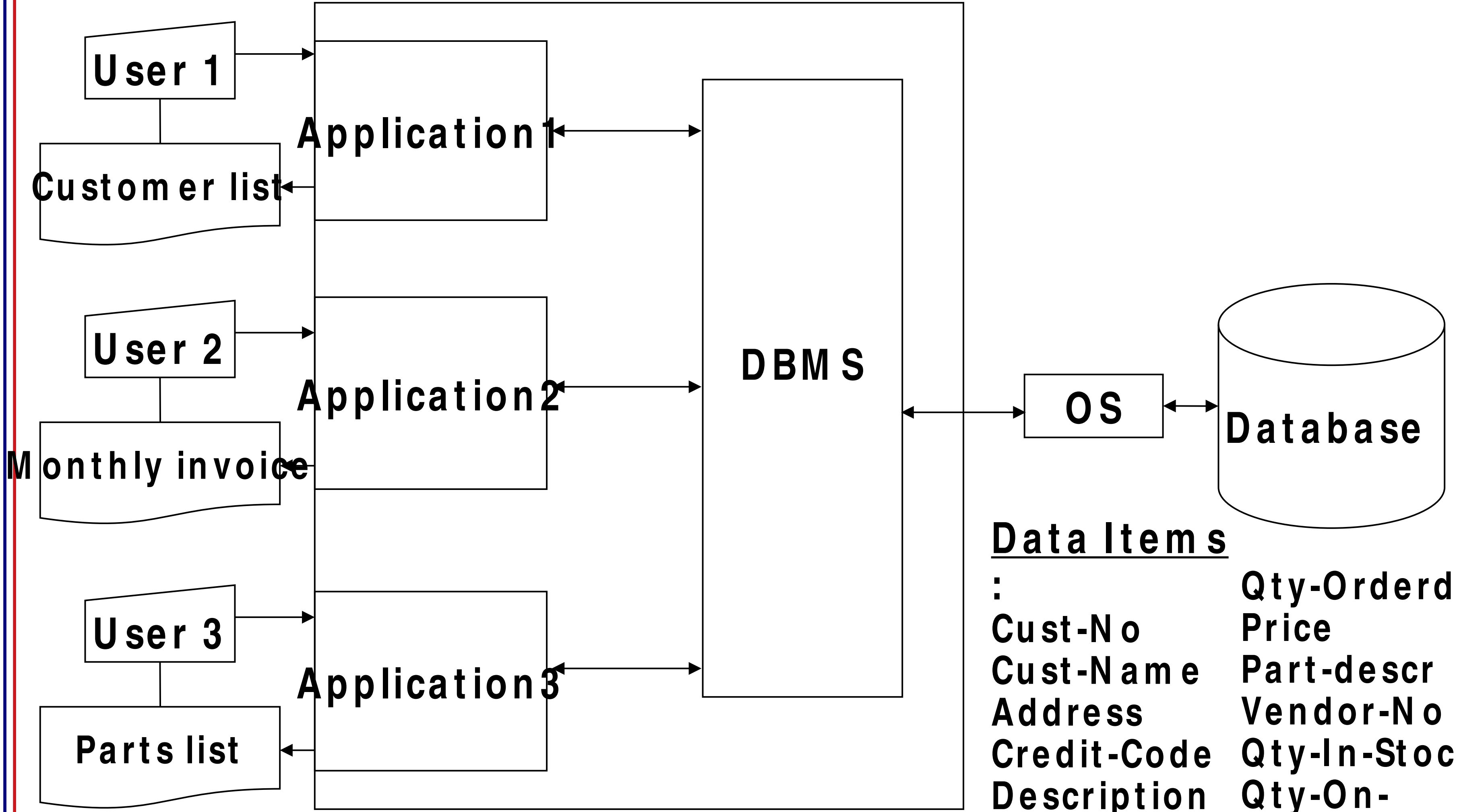
Redundancy of data: Same data is duplicated perhaps in slightly different format over various files

Multiple updates: Changes have to be made to all files containing the same data. ***Possibility of inconsistency***

Waste of storage space:

Reliability and better local control


Naïve User
Casual Users
Web User
Application
Application
. . .
Application
DBA
Database Management System
Online storage
Online storage
Online storage



Data Items:

Cust-No
Cust-Name
Address
Credit-Code
Description
Part-No
Qty-Orderd
Price
Part-descr
Vendor-No
Qty-In-Stock
Qty-On-Order

Pros & Cons of DBMS

Reduce data redundancy and avoiding inconsistency

Provide Concurrent access

Offer Centralized control

- security(appropriate authorization and its control),
- integrity(constraints and their enforcements)
- reliability(backups and replication)

Data abstraction and independence

First Step: Data Models

- Data Model: concept to describe data
- Schema: description of a collection of data using a specific data model
- Relational Model: Based on the concept of **relation***(table with rows and columns)*

A Data Model is a collection of concepts for describing

- Entities(objects) and relationships among them
- Expressing the semantics and constraints from the real world

Object-Based Modeling Techniques

- Entity-Relationship (ER) Model
- Object-Oriented (OO) Model

Record-Based Models

- Hierarchical Model: used by earliest DBMS – IBM's IMS
- Network Model: second generation DBMS - DBTG
- Relational Model: the first based on theory - relations
  (RA, RC, Datalog)

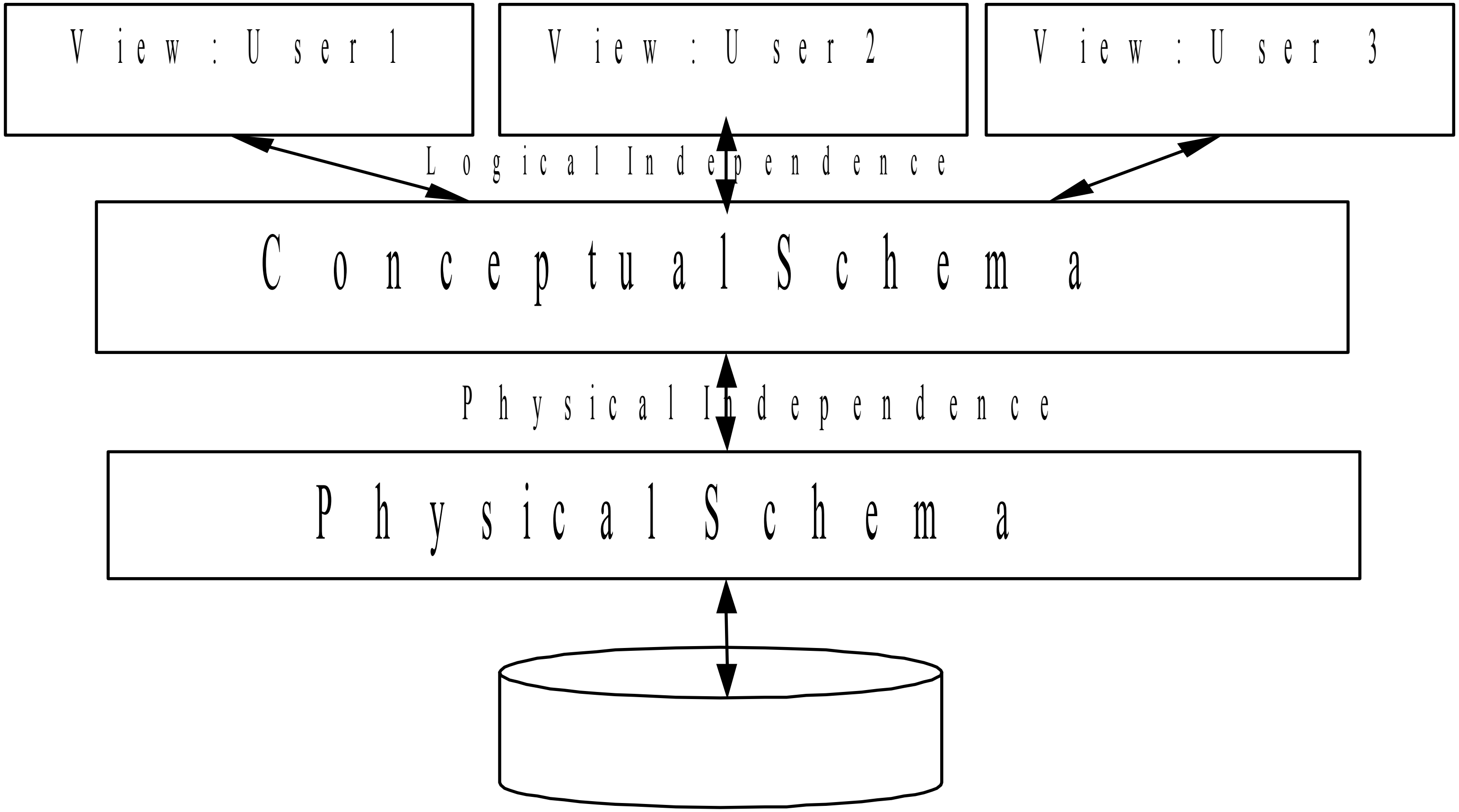
Employee Name
Employee Phone Number

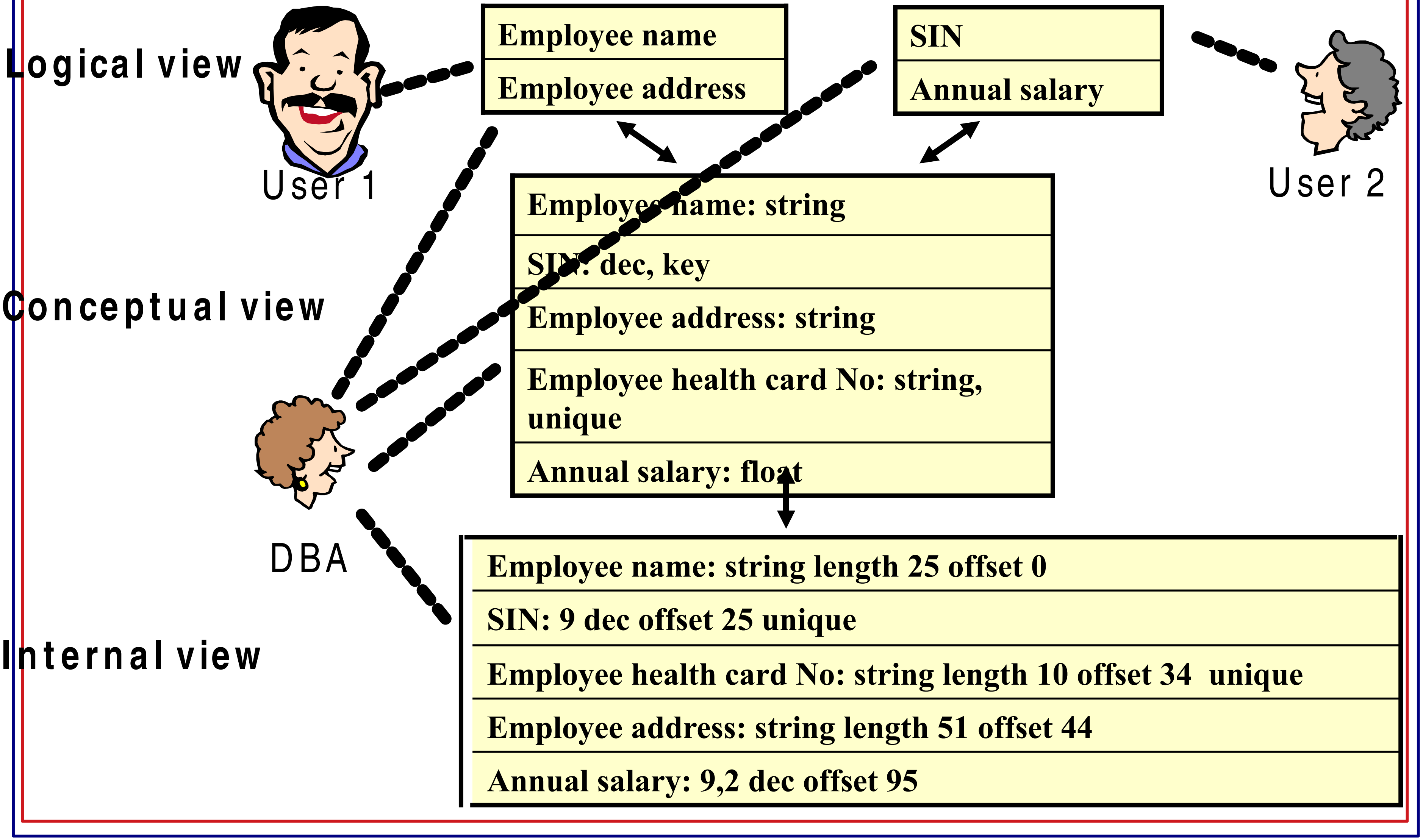
Employee Name
Employee SIN
Employee Salary

Employee Name
Employee Phone Number
Employee SIN
Employee Address
Employee Annual Salary
Employee YTD Salary

Employee Name string
Employee Phone Number digits
Employee SIN digits
Employee Address string
Employee Annual Salary money units
Employee YTD Salary money units

Three level Concepts

- 



Logical view

User 1

Employee name

Employee address

SIN

Annual salary

User 2

Conceptual view

Employee name: string

SIN: dec, key

Employee address: string

Employee health card No: string, unique

Annual salary: float

DBA

Internal view

Employee name: string length 25 offset 0

SIN: 9 dec offset 25 unique

Employee health card No: string length 10 offset 34 unique

Employee address: string length 51 offset 44

Annual salary: 9,2 dec offset 95

Three levels & Independence

- User View: How users view data - derived from conceptual view-
- Conceptual Schema: Logical structure of the database
- Physical Schema: The actual files and indices used
- Schema defined using DDL

Data Independence: modify definition of schema at one level without affecting a schema definition at a higher level.

Logical Data Independence: modify logical schema without causing application programs to be rewritten

adding new fields to a record or changing the type of a field

Physical Data Independence: modify physical schema without causing logical schema or applications to be rewritten

changing file structure from sequential to direct access

University Database

- External Schema:

Course_Enrol(C#:char, Number:int);

- Conceptual Schema:

Student(S#, Name, Dept)

Course(C#, Cname, Credits)

Enrollment(C#, S#, grade)

- Physical Schema:

files, indexed on S#, C#, etc

A database schema is a description of a particular collection of data, using a given data model

Part of a schema for a university. database in relation model would contain among others, the following:

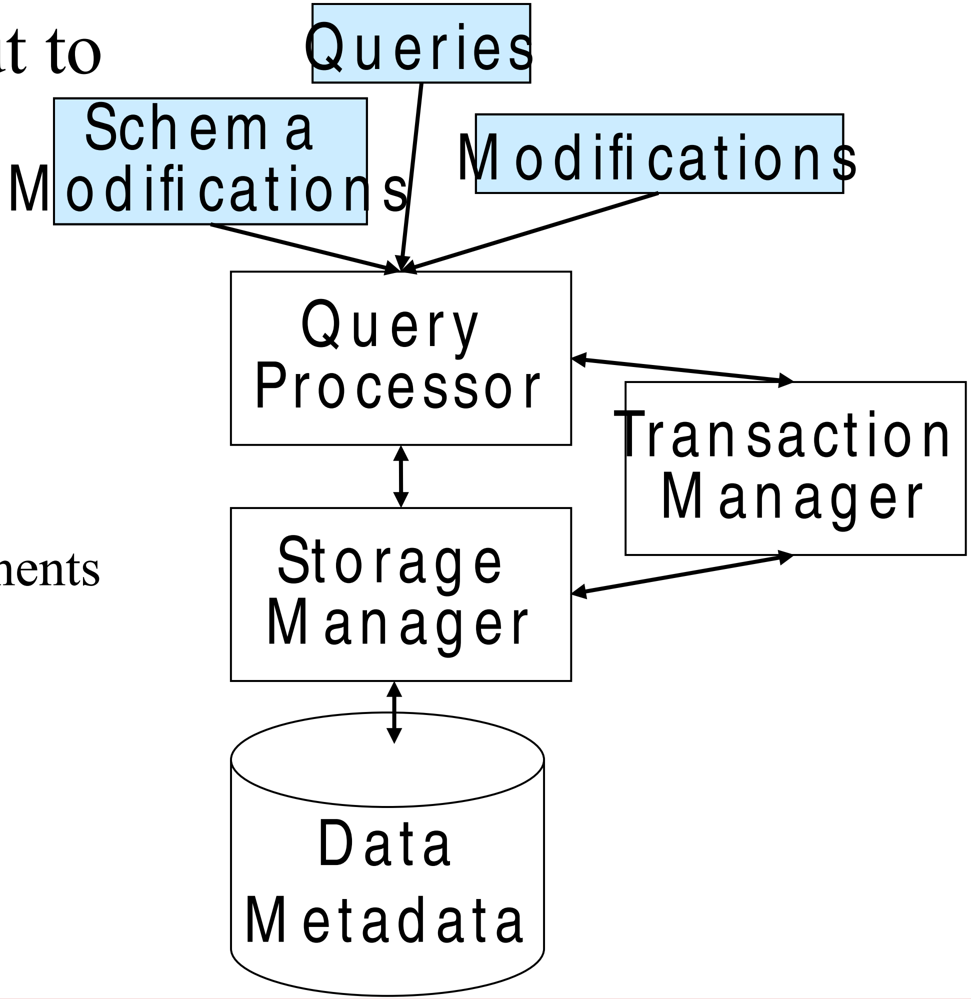
Students (<u>sid</u>, name, department, dob, address)

An instance of a database schema is the actual content of the database at a particular point in time

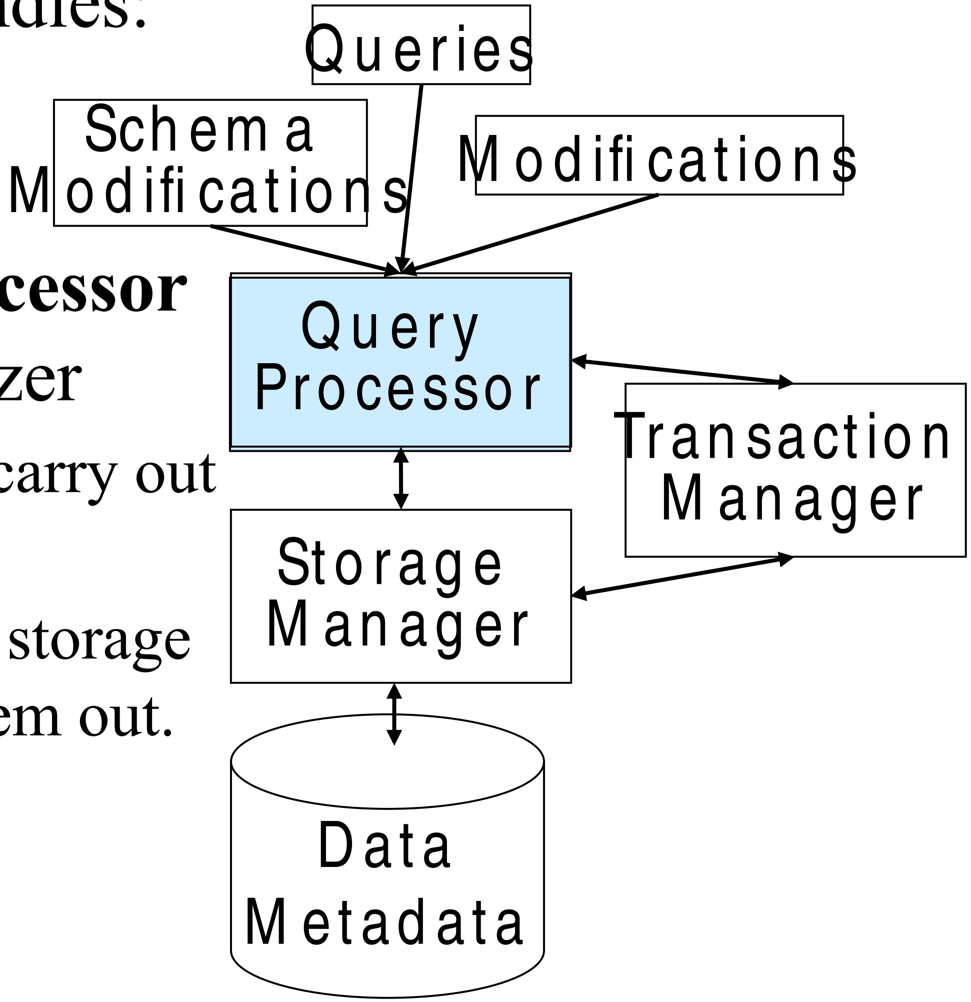
sid	name	department	dob	address
1112223	John Smith	CS	12-01-82	22 Pine, #1203
2223334	Ali Brown	EE	31-08-73	2000 St. Marc
3334445	Youwong Li	CS	23-11-79	1150 Guy

The Architecture of a DBMS

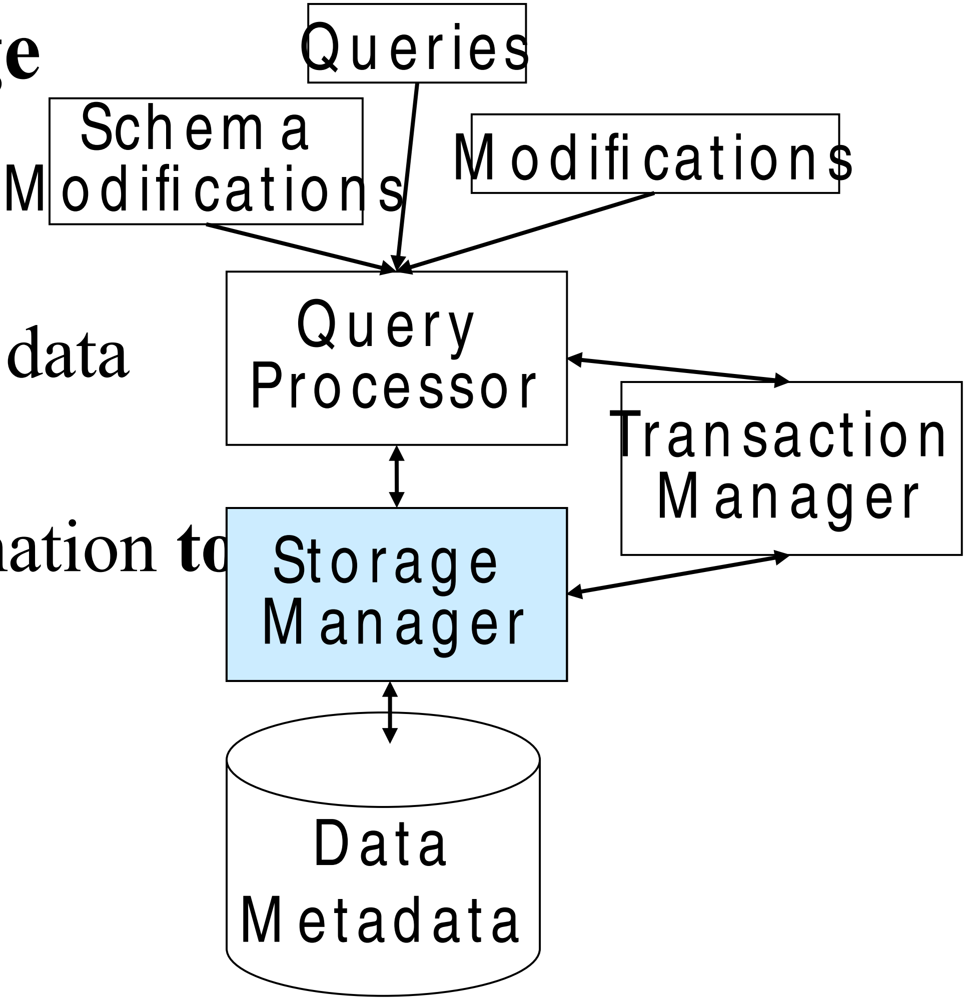
- There are 3 types of input to DBMS:
  - Access via queries
  - Updates to data
  - Updates to model
    - □ Initial database creation,
    - □ addition to schema components
    - □ schema modifications



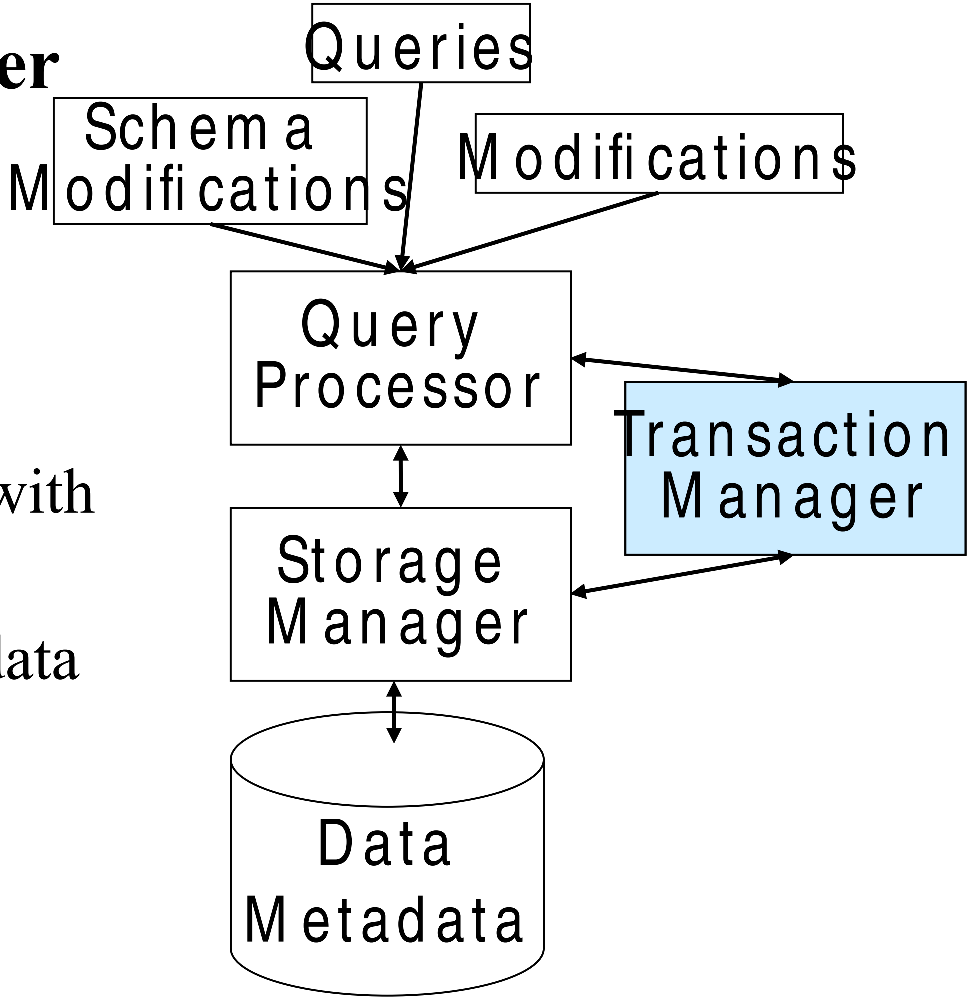
- The **query processor** handles:
  - Queries
  - Updates
- The job of the **query processor** which includes an optimizer
  - To find the “best” way to carry out a requested operation
  - To issue commands to the storage manager that will carry them out.



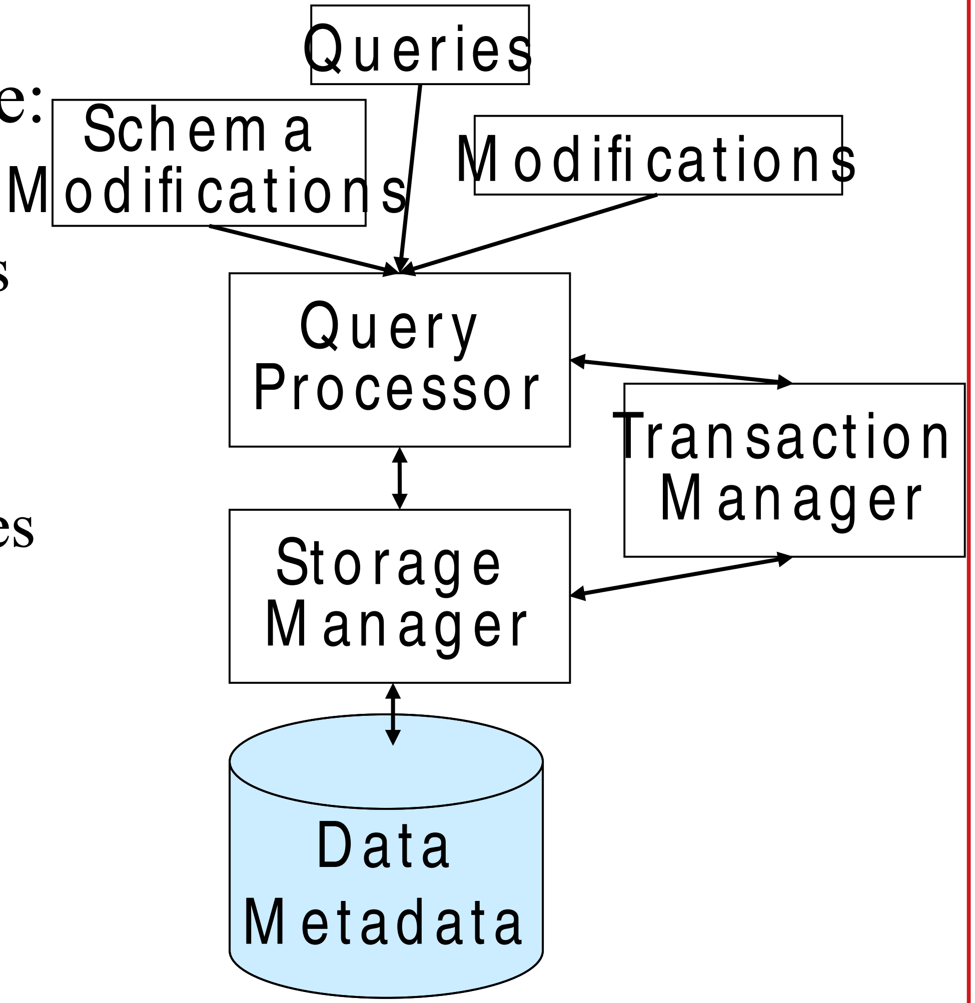
- The job of the **storage manager** is
  - To obtain requested information **from** the data storage
  - To modify the information **to** the data storage when requested.



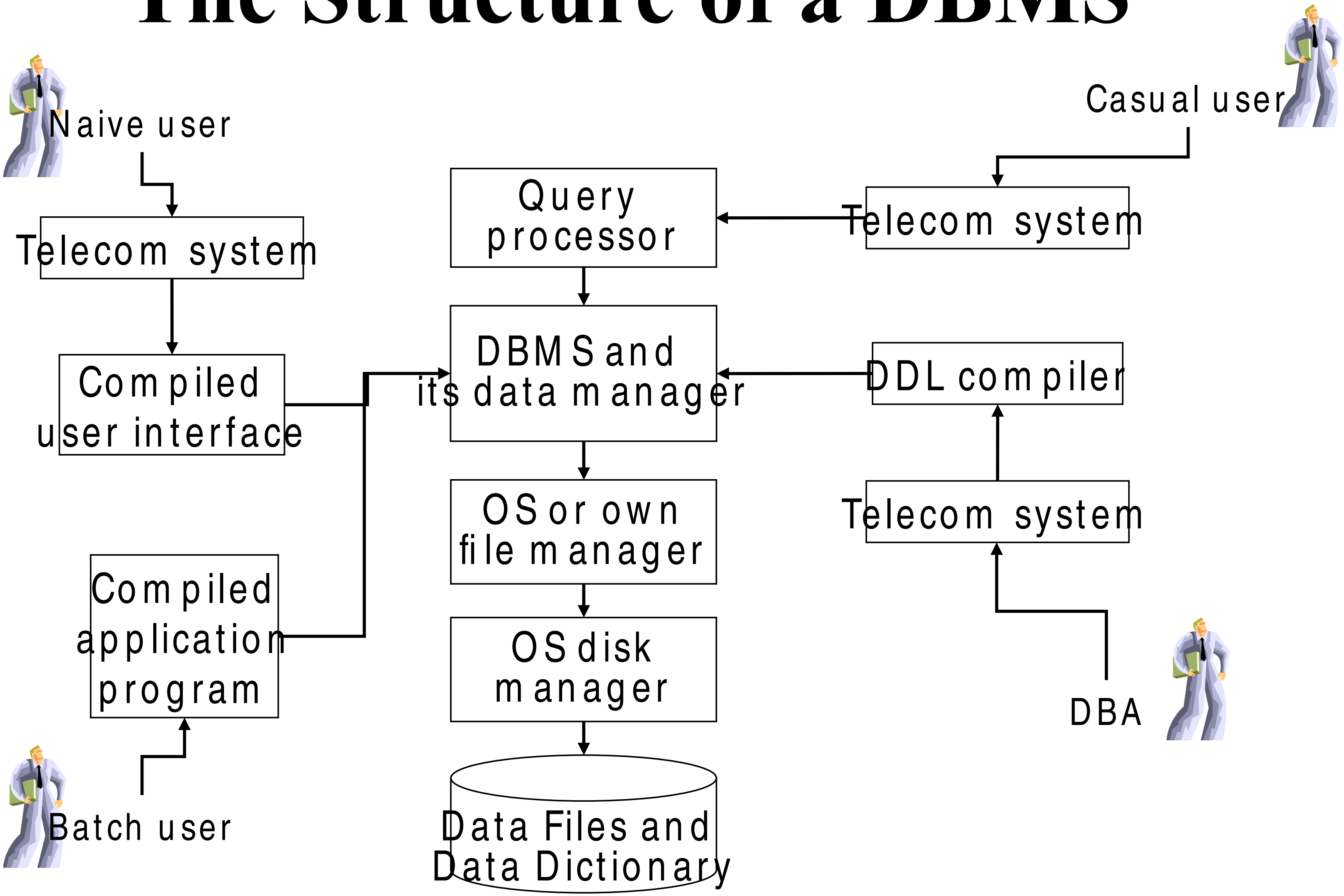
- The **transaction manager** is responsible for the **enforcing ACIDity**
  - several concurrent transactions(one or more queries) do not interfere with each other
  - the system will not lose data even if there are failures (*done through Recovery subsystem*)



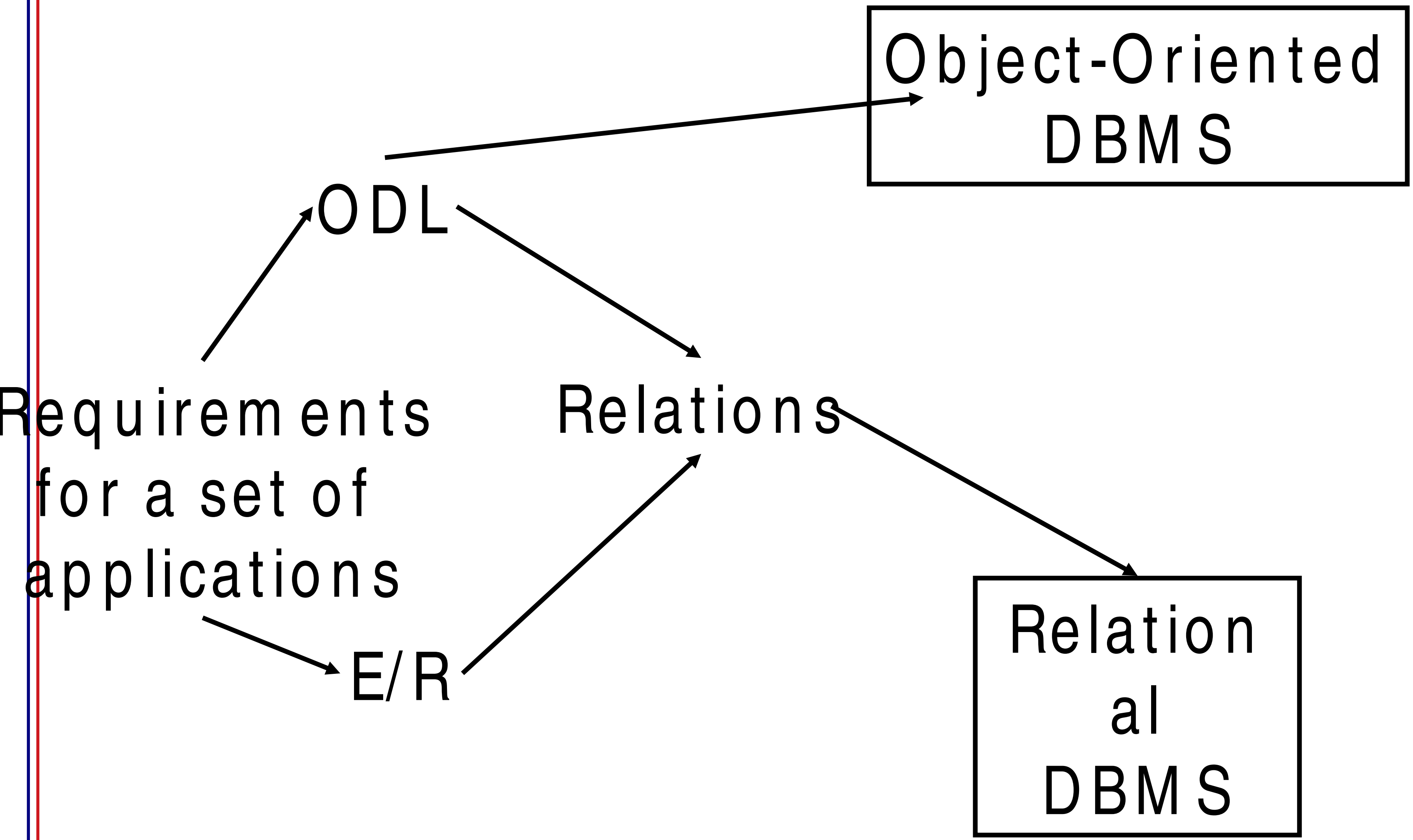
- Database contents include:
  - Metadata for the DBMS and one or more databases
  - Data belonging to one or more databases
  - Access aids such as indices and statistics



The Structure of a DBMS



Database Design Process
and
Conceptual Design



Requirements for a set of applications

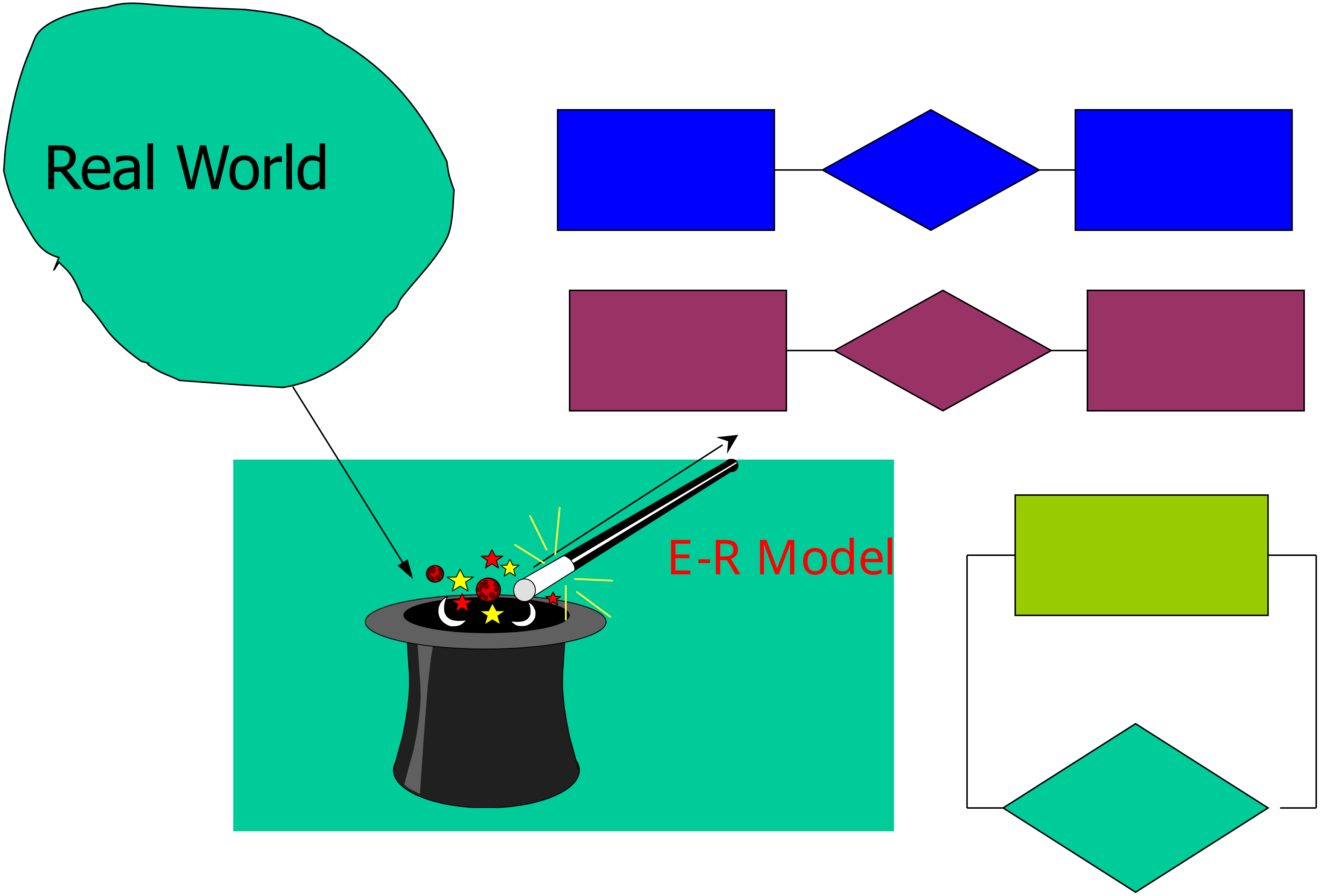
ODL

E/R

Relations

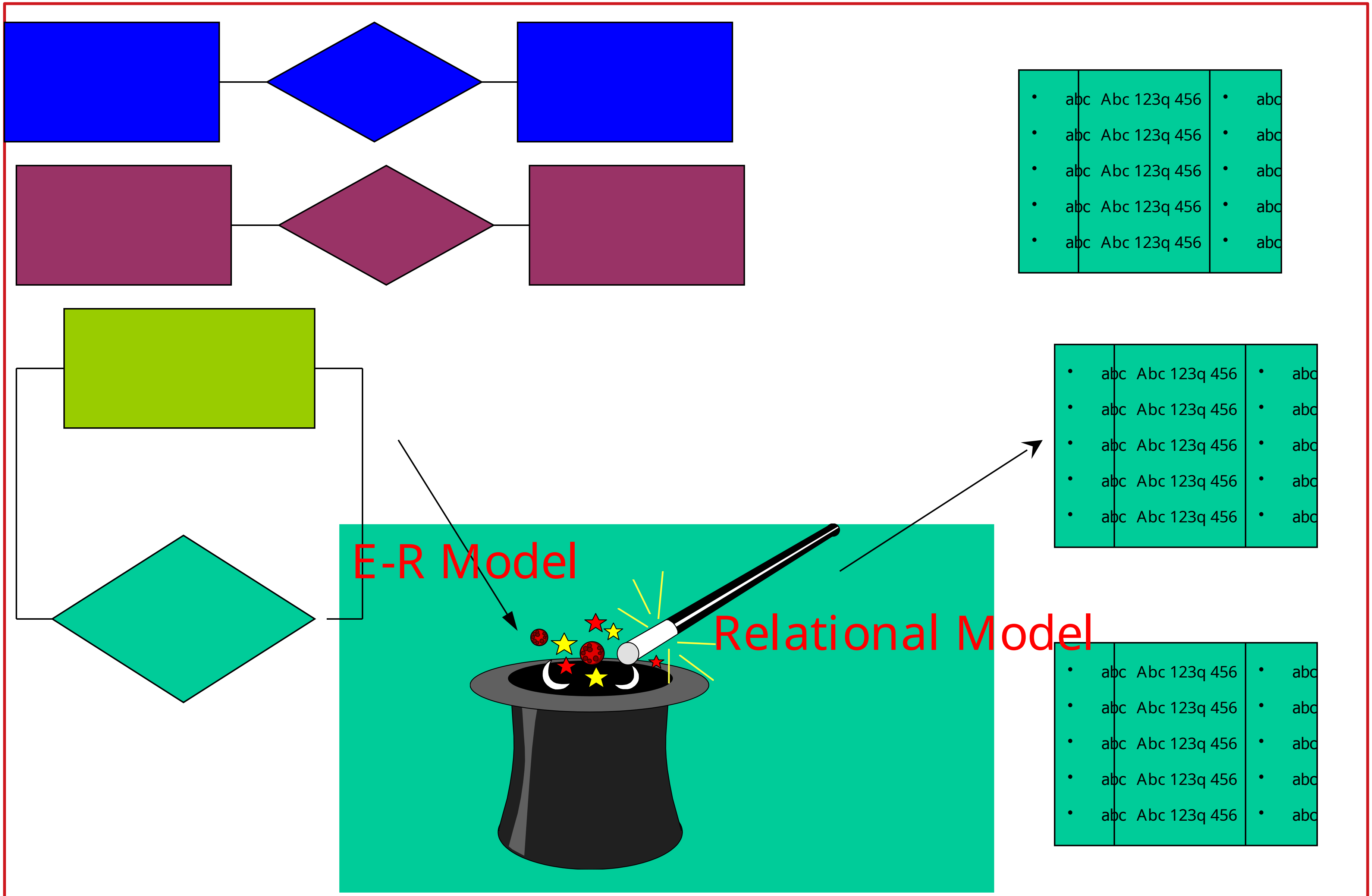
Object-Oriented DBMS

Relational DBMS



Real World

E-R Model



E-R Model

Relational Model

•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc

•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc

•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc
•	abc	Abc 123q 456	• abc

Relational Model

In this model, the data is organized in relations (tables)

Relational database schema: DDL component of SQL

set of table names

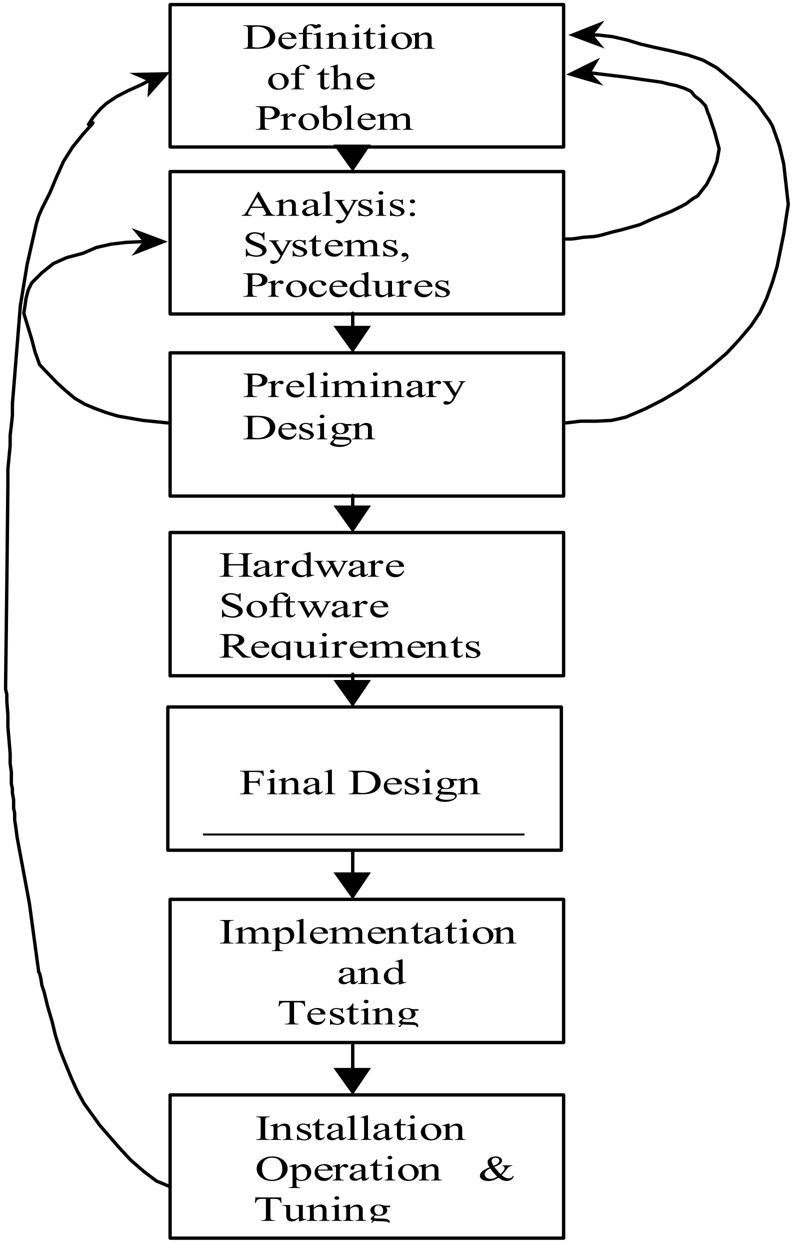
list of attributes for each table and their properties

Examples of tables from a university database:

Student : stud_number, name, address, program

Department: name, budget_code, room, phone

Course: name, number, credits



Definition of the Problem

Analysis: Systems, Procedures

Preliminary Design

Hardware Software Requirements

Final Design

Implementation and Testing

Installation Operation & Tuning

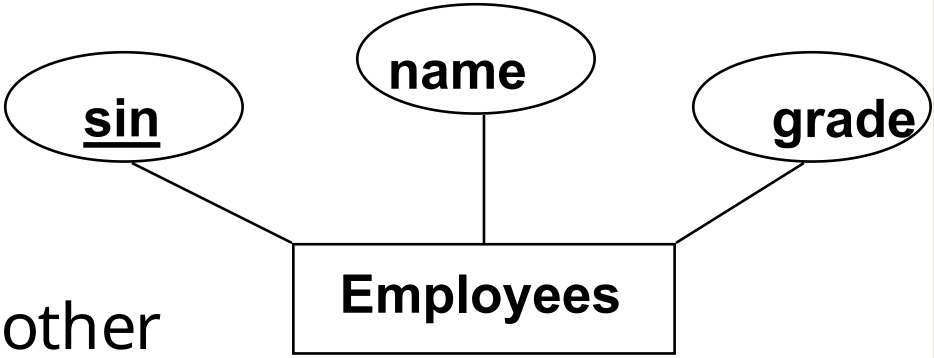
Database Design Process

- *Definition of the problem*
- *Study underlying applications(Procedure Manuals, Interviews etc.)*
  - What are the *entities* and *relationships* involved?
  - What details about them should be in the database?
  - What are the *procedures, business rules, constraints*?
  - Who are the users? What do they need?
- *Preliminary Conceptual design*:
  - ER Model

Database Design Process

- □ *Software/Hardware Requirements*
  - ✚ UML for software design
- □ *Final Design: Schema Refinement: (Normalization)*
  - ✚ Check relational schema for redundancies and related anomalies.
  - ✚ External Schemas, indices, views, access methods
- □ *Application programs, forms, reports, user interfaces*
- □ *Implementation and testing*
- □ *Installation and Tuning:*
  - ✚ Data Distribution, Physical re-design
  - ✚ Performance, Security, Backup & Recovery.

ER Model



- *Entity*: Real-world object distinguishable from other objects.
  - An entity is described using a set of *attributes.*
- *Entity Set*: A collection of similar entities.
  - All entities in an entity set have the same set of attributes.
  - Each entity set has a *key.*
  - Each attribute has a *domain.*
  - Can map entity set to a relation easily.

```
CREATE TABLE Employees
    (sin CHAR(9),
    name CHAR(25),
    grade INTEGER,
    PRIMARY KEY (sin))
```

```
mysql> CREATE TABLE Employees
              (sin CHAR(9),
       name CHAR(25),
       grade INTEGER,
       PRIMARY KEY  (sin));
Query OK, 0 rows affected (0.00 sec)
mysql> show tables;
+-----------------+
| Tables_in_db11s |
+-----------------+
| Employees       |
+-----------------+
mysql> desc Employees;
+-------+----------+------+-----+---------+-------+
| Field | Type     | Null | Key | Default | Extra |
+-------+----------+------+-----+---------+-------+
| sin   | char(9)  | NO   | PRI |         |       |
| name  | char(25) | YES  |     | NULL    |       |
| grade | int(11)  | YES  |     | NULL    |       |
+-------+----------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

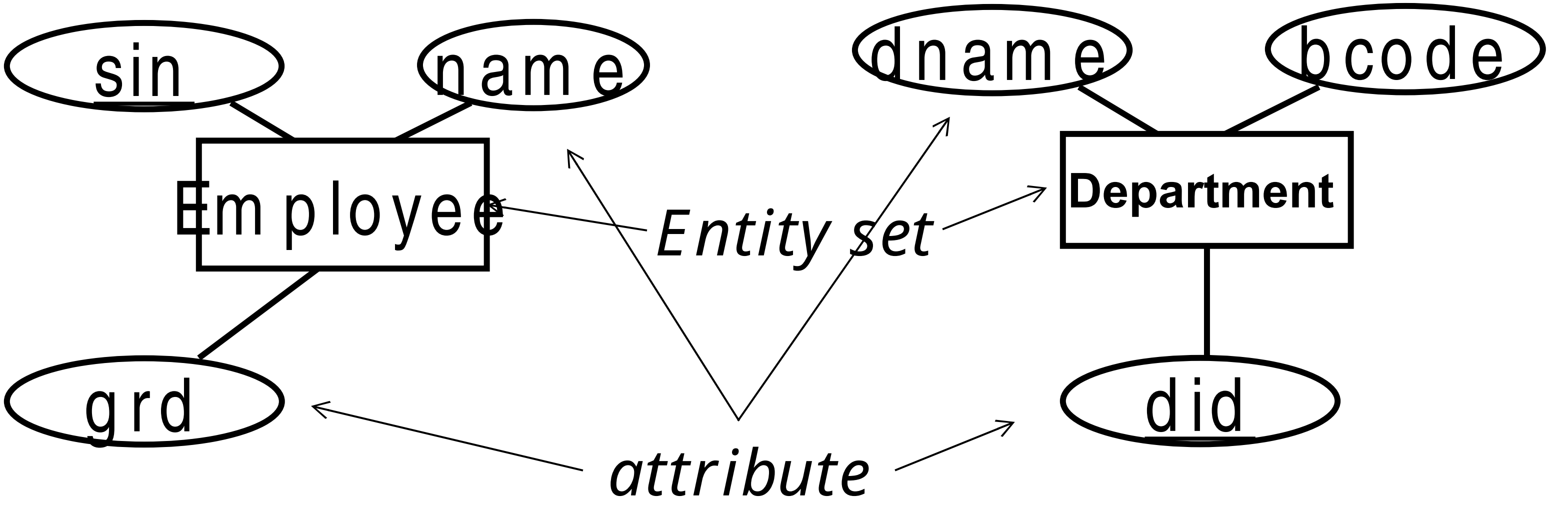
Note: size of integer is defaulted to 11

The Extra field contains any additional information that is available about a given column.
The value is auto_increment for columns that have the AUTO_INCREMENT attribute and empty otherwise.

```
CREATE TABLE Department
    (did mediumint not null  auto_increment,
     dname CHAR(16),
     bcode char(12),
     PRIMARY KEY  (did));
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> desc Department;
+-------+--------------+------+-----+---------+----------------+
| Field | Type         | Null | Key | Default | Extra          |
+-------+--------------+------+-----+---------+----------------+
| did   | mediumint(9) | NO   | PRI | NULL    | auto_increment |
| dname | char(16)     | YES  |     | NULL    |                |
| bcode | char(12)     | YES  |     | NULL    |                |
+-------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)
```

Entities and entity sets



All employees, and departments have the same set of properties(attributes)

To distinguish one instance of an entity in an entity set from others, we introduce an identifying attribute

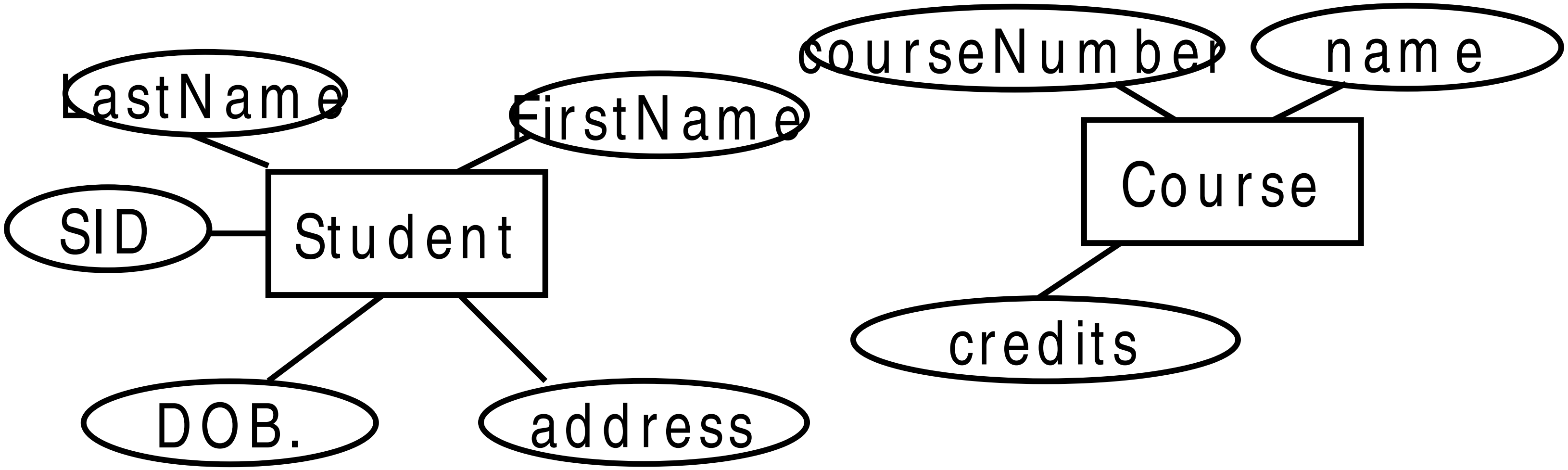
This is the primary key and it is underlined

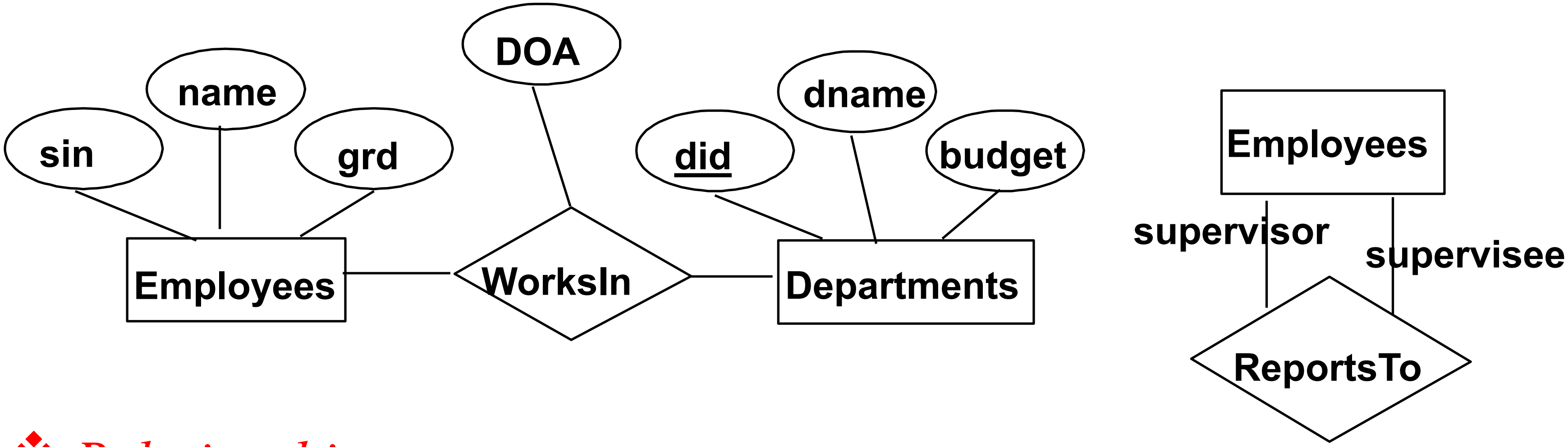
Entity – Real world object distinguishable from other objects of the same type

Entity Set -- A collection of similar entities: all have same set of properties

ODL:

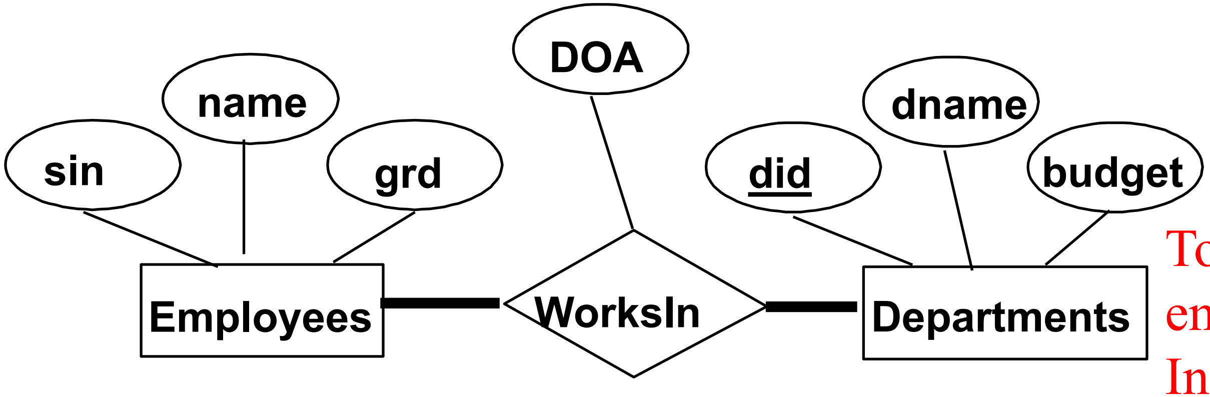
Object corresponds to entity **Class** corresponds to entity set



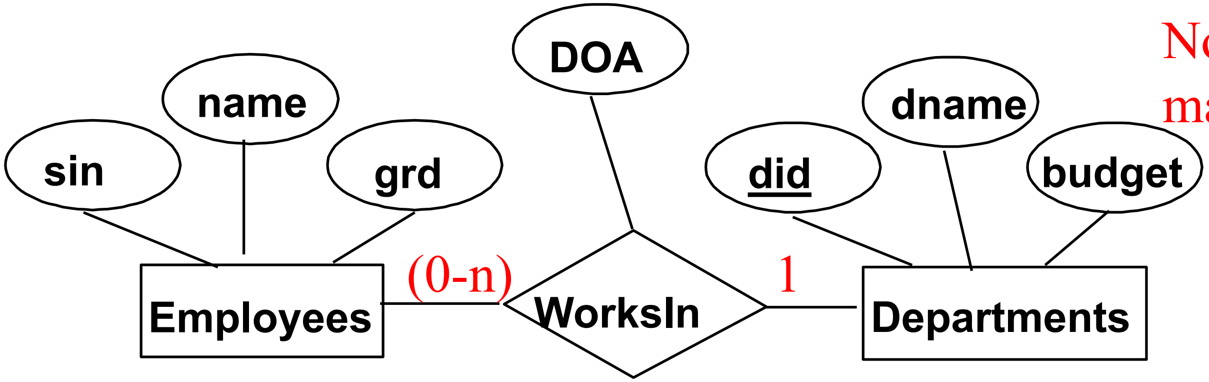


- *Relationship*:
  - Association among 2 or more entities.
- *Relationship Set*: Collection of similar relationships.
  - An n-ary relationship set R expresses an association among n entity sets E1 ... En; each relationship in R involves entities e1 ∈ E1, ..., en ∈ En

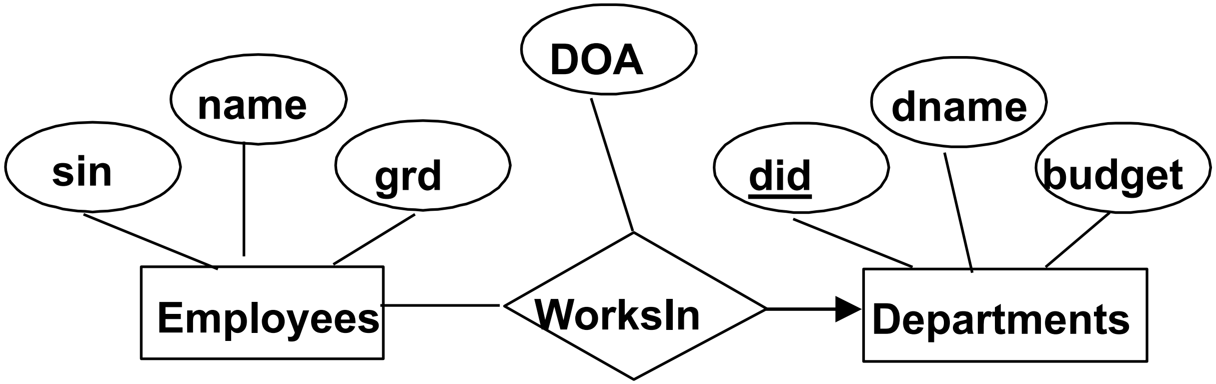
Same entity set could participate in different relationship sets, or in different "roles" in same set.



Total participation of all employees & departments In the WorksIn relationship No Employee or Department may exist without being related

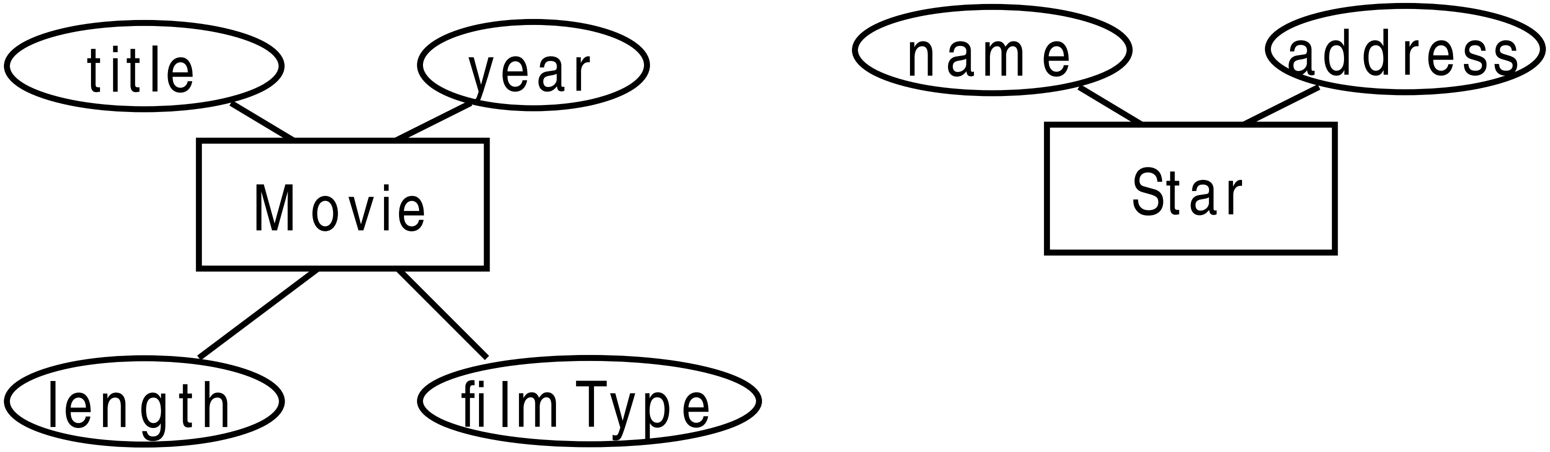


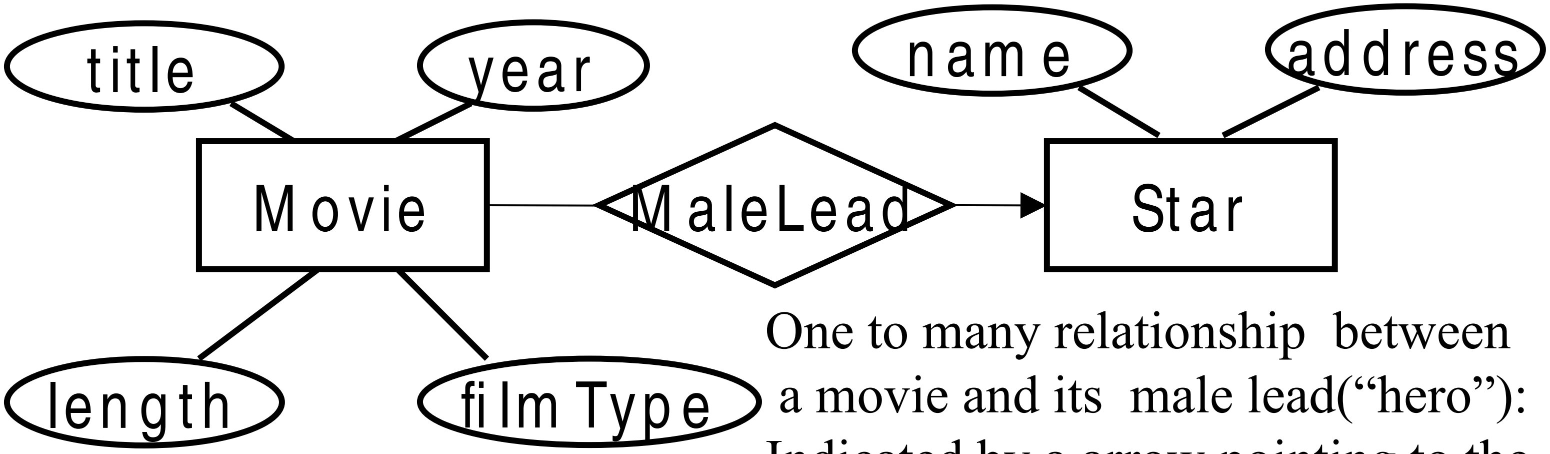
Many to one relationship: many employees in a department; but an employee is assigned to only one department



Alternate way of showing a many-to-one relationship

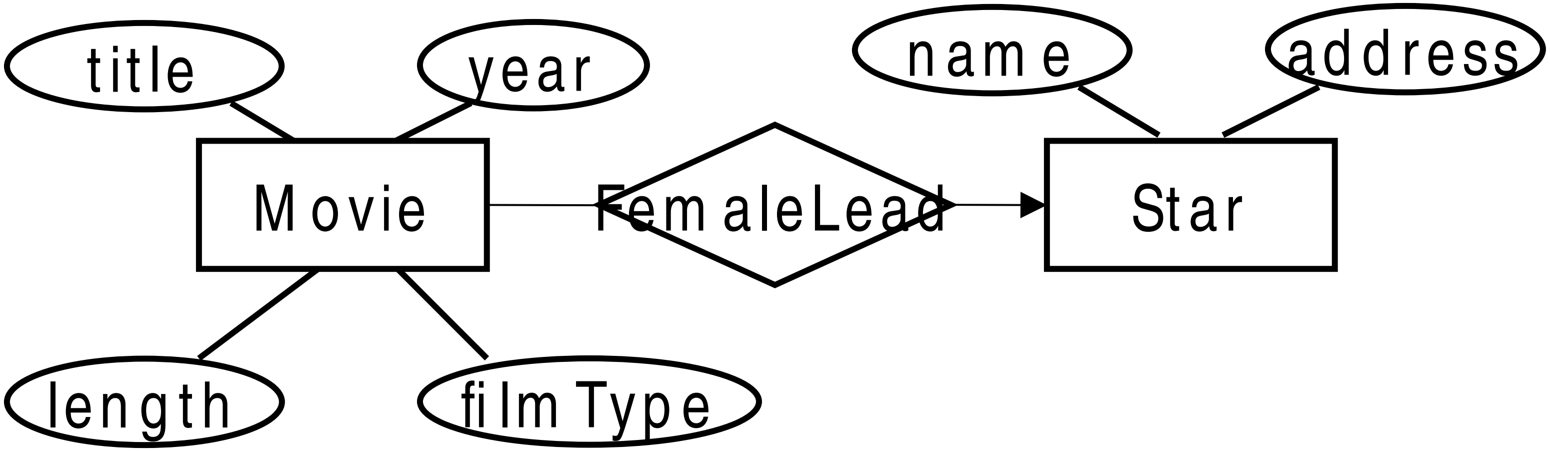
Entities and entity sets

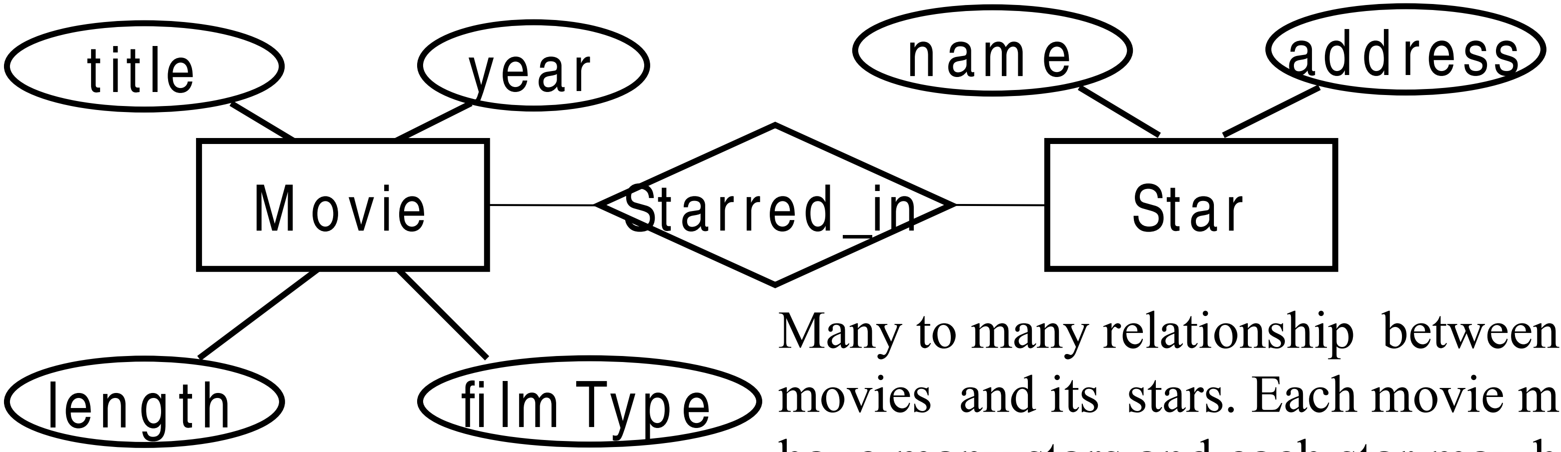




One to many relationship between a movie and its male lead("hero"): Indicated by a arrow pointing to the "one side" – **A movie has but one main role, The star may be a lead in many movies**

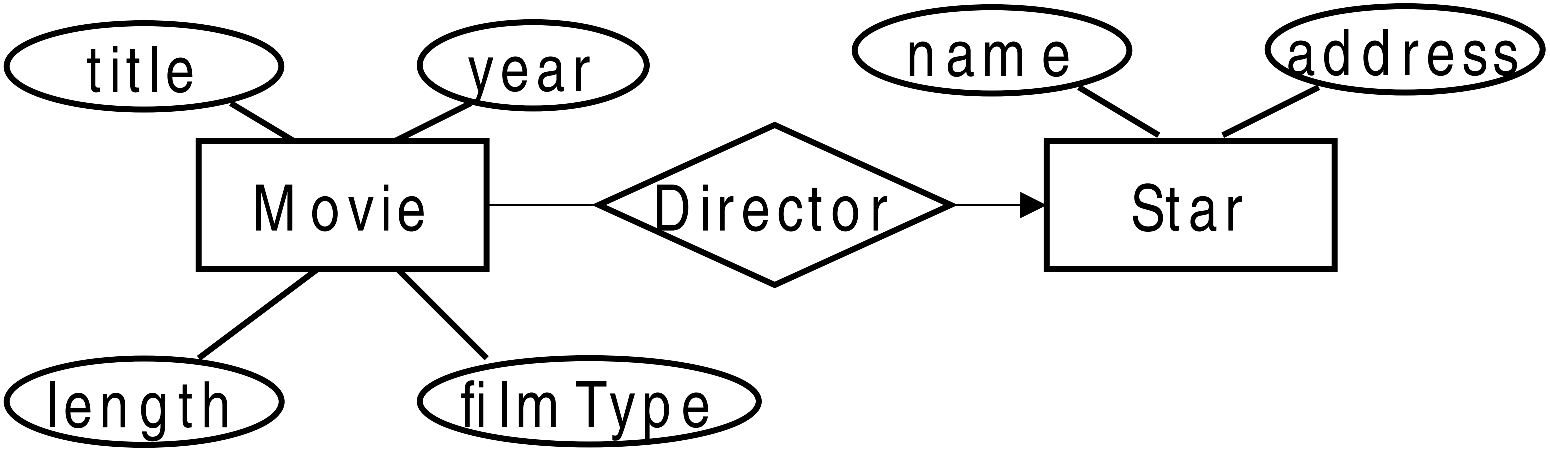
Entities and entity sets





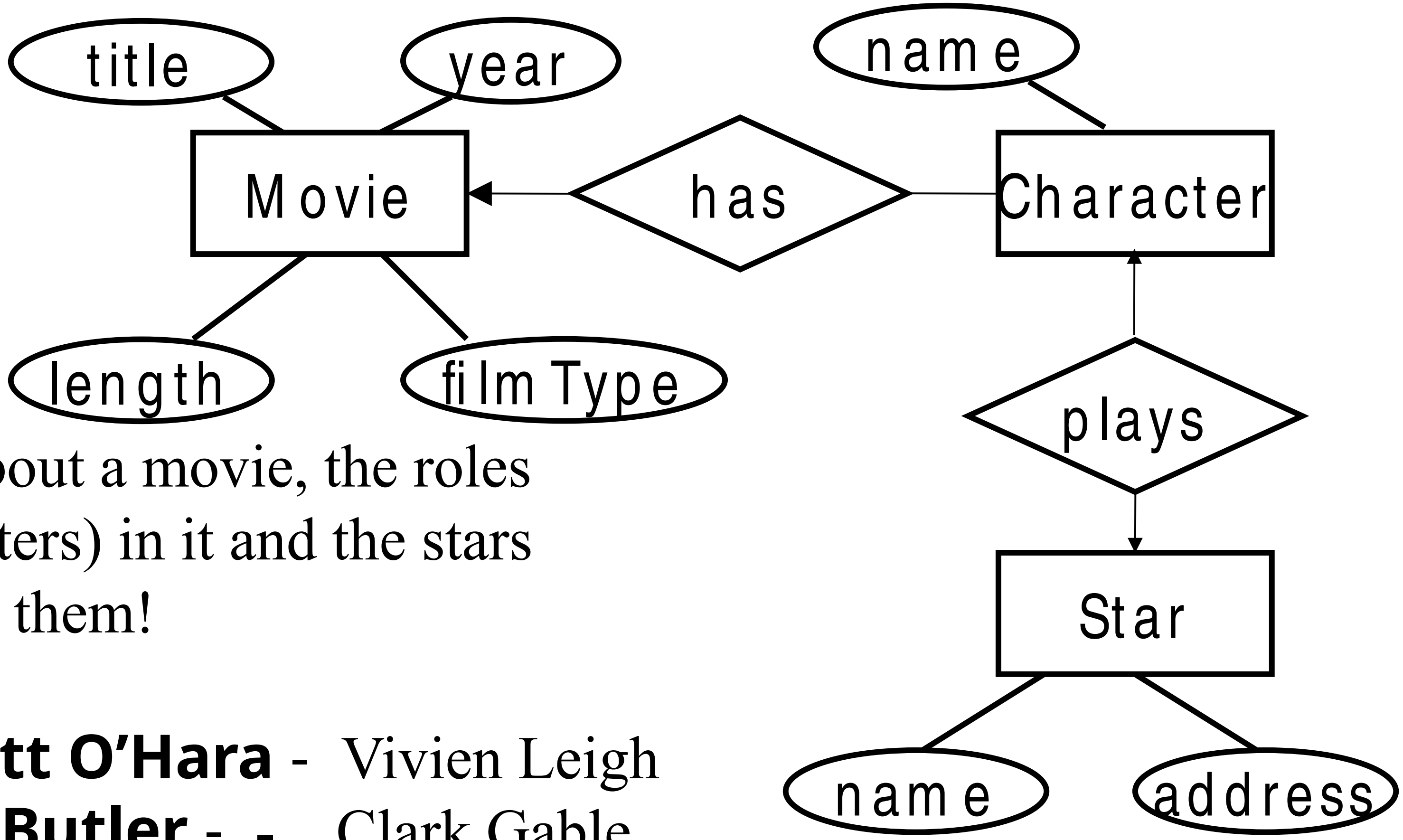
Many to many relationship between movies and its stars. Each movie may have many stars and each star may have featured in many movies. Indicated by no arrows on the connecting lines.

Entities and entity sets



How about a movie and the roles(characters) in it and the stars playing them!

Entities and entity sets



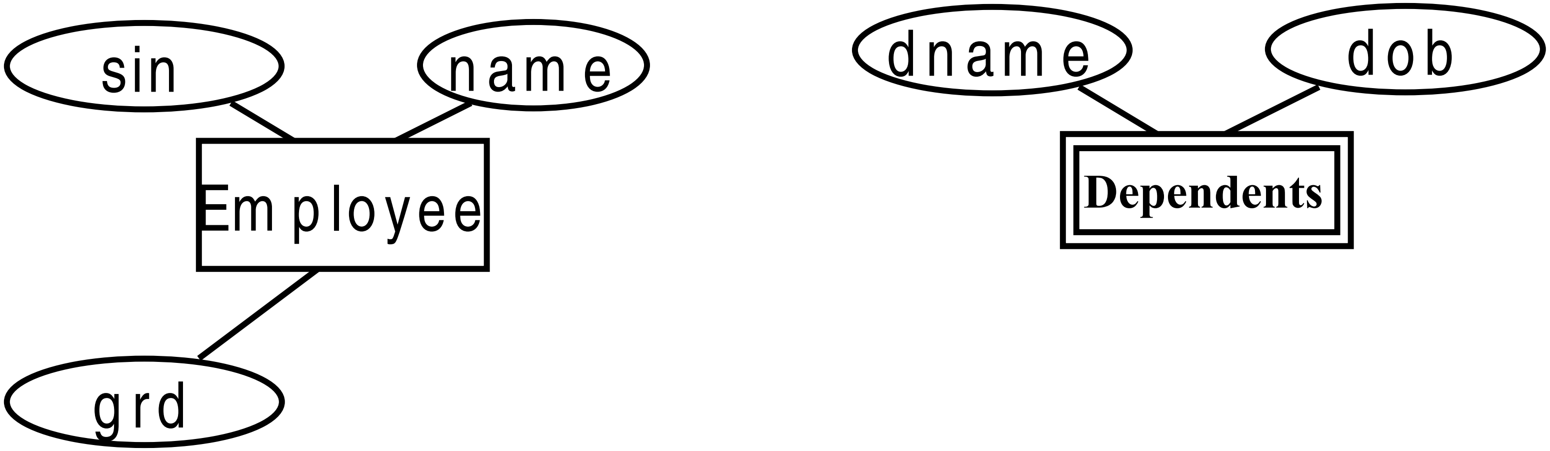
How about a movie, the roles (characters) in it and the stars playing them!

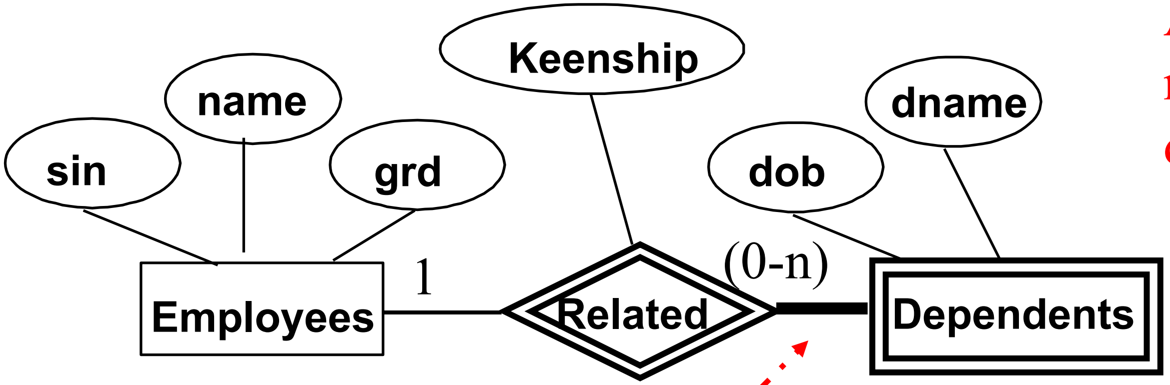
Scarlett O’Hara - Vivien Leigh
Rhett Butler - - Clark Gable

Alex Guinness plays eight members of the D'Ascoyne family in Kind hearts and coronets(1949)

Matt Damon played the lead in the *Bourne* triology.

Entities and entity sets





An employee may have 0 to n dependents

Total participation

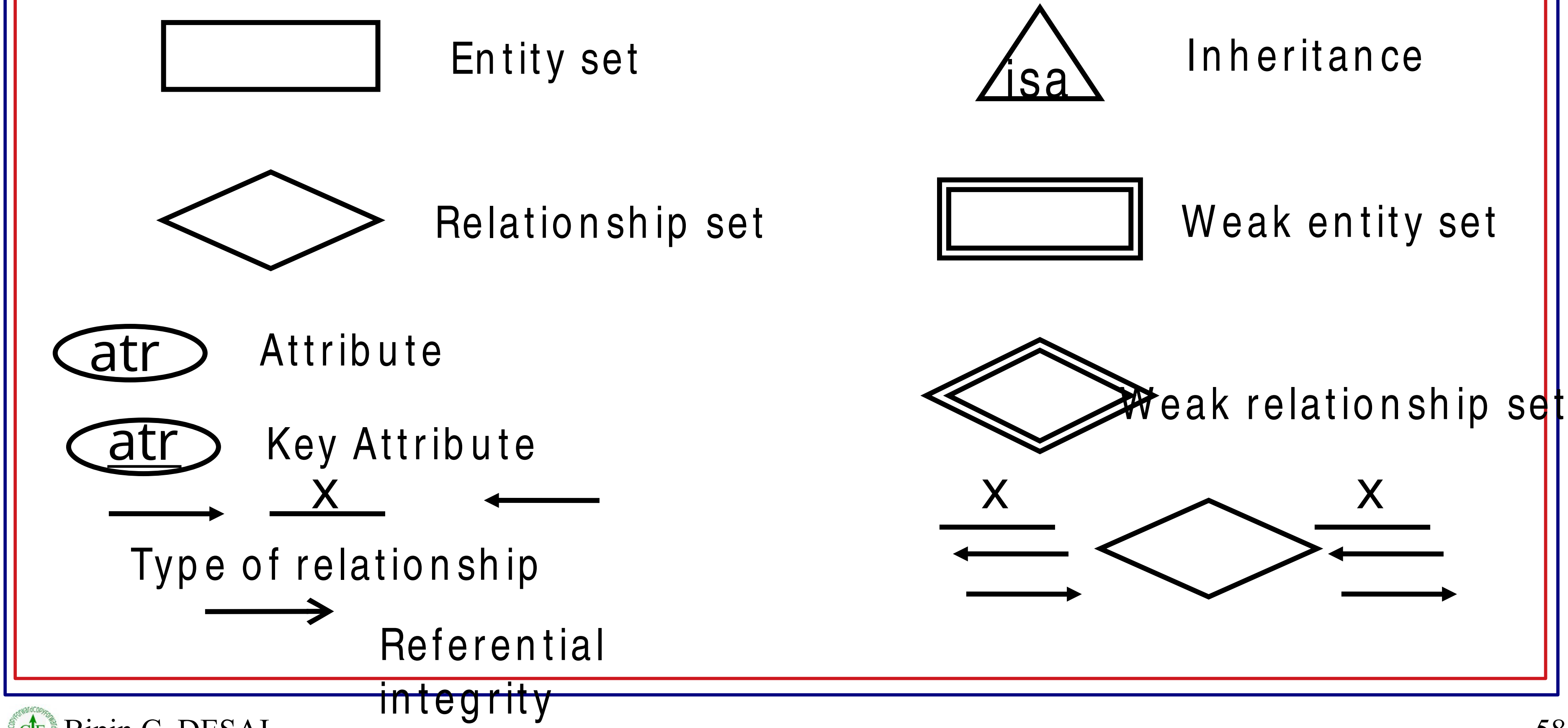
All dependents must be related to some employee(but only one!)

E/R model is a *graphical* approach to database modeling

E/R is widely used in database design

E/R model grew out of modeling application database

No standard for E/R diagrams: a number of variations



Entity set

Inheritance

Relationship set

Weak entity set

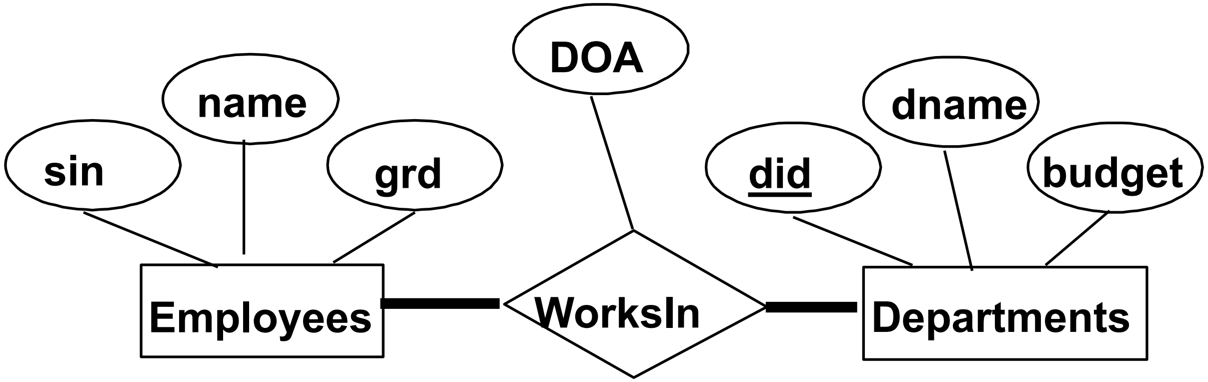
Attribute

Weak relationship set

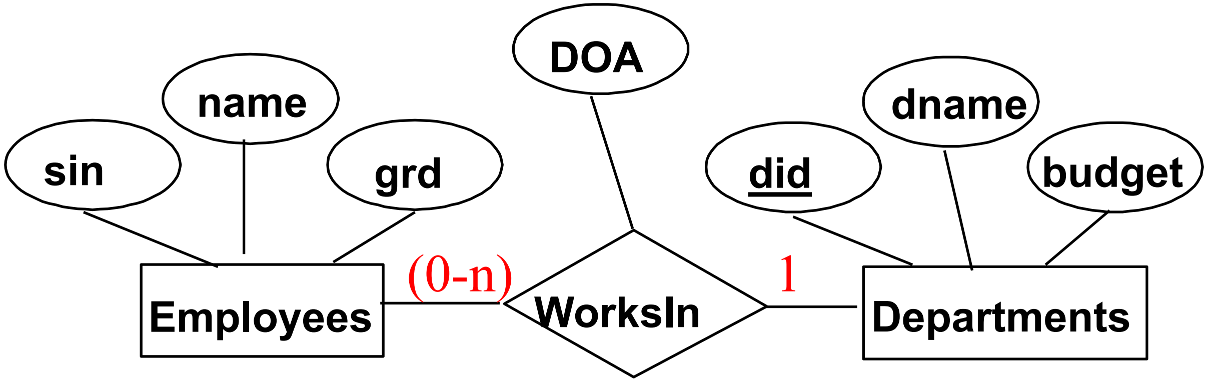
Key Attribute

Type of relationship

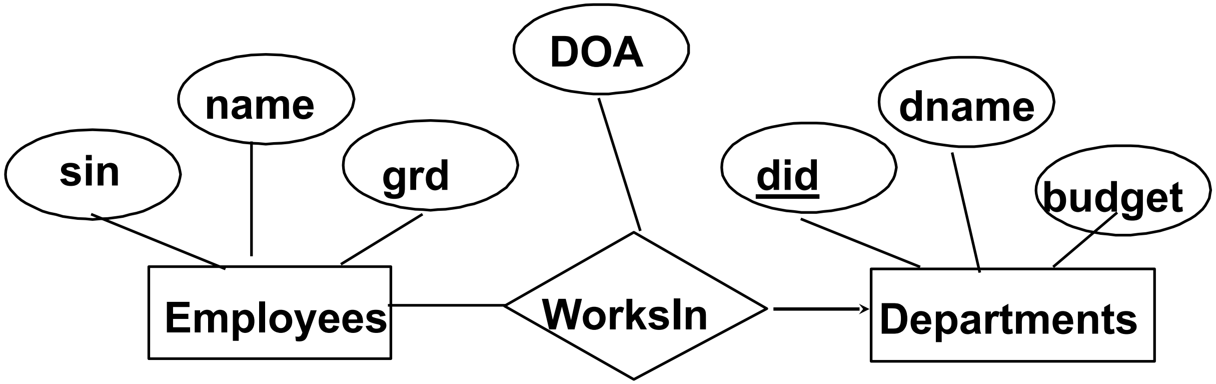
Referential integrity



Total participation of all employees & departments In the WorksIn relationship

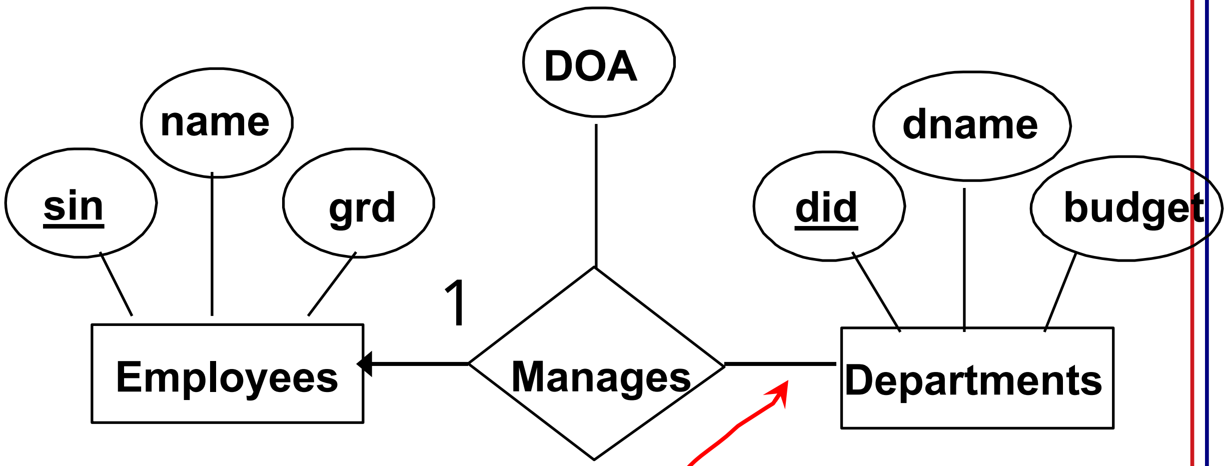


Many to one relationship: many employees in a department;
but an employee is assigned to only one department



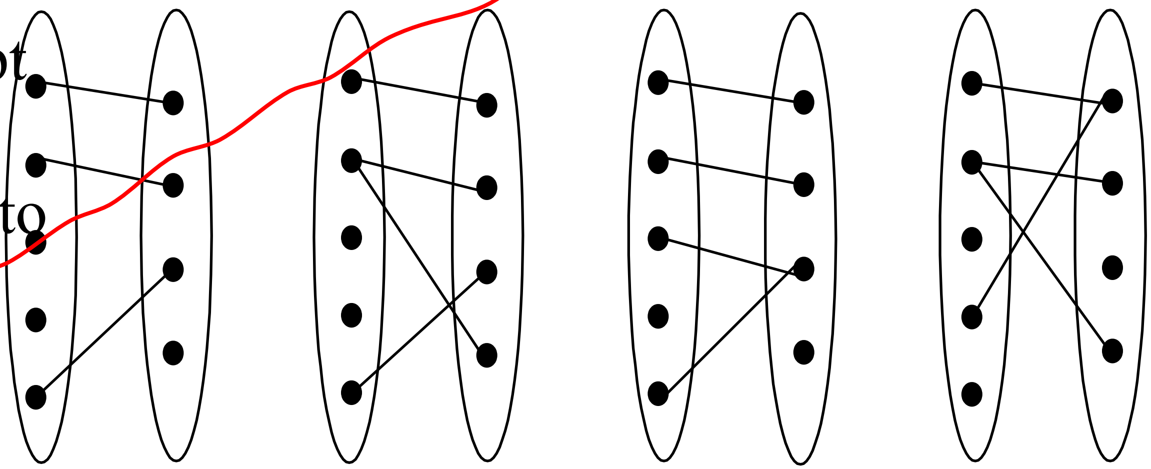
Alternate way of showing a many-to-one relationship

Key Constraints



- Consider Works_In: An employee can work in many departments; a dept can have many employees.
- In contrast, each dept has at most one manager, according to the *key constraint* on Manages.

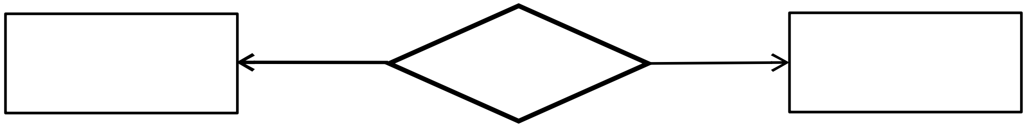
A department can have only one manager,
an employee could manage many departments.



- Relationship sets can have *attributes*
- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
    - This set of attributes forms **superkey** for the relation.
  - All descriptive attributes.

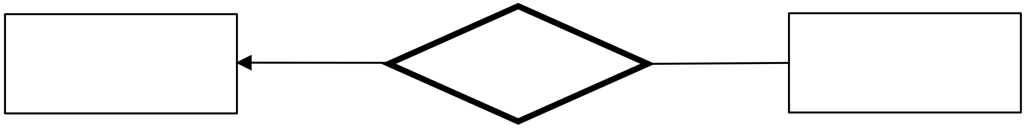
```
CREATE TABLE WorksIn
 (sin  CHAR(9),
 did  INTEGER,
 DOA  DATE,
 PRIMARY KEY (sin, did),
 FOREIGN KEY (sin)
       REFERENCES Employees,
 FOREIGN KEY (did)
       REFERENCES Departments)
```

If a relationship is 1-to-1 primary key is from *either* of the ‘1’ side, the other side is a *foreign* key



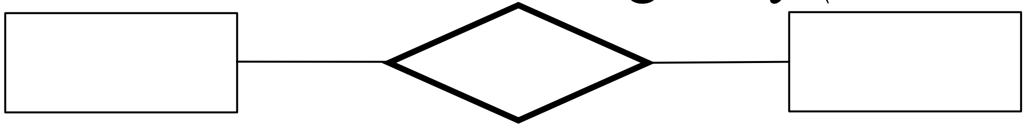
If a binary relationship between two entity sets is 1-to-1,

- the primary key of the relationship is the key of the entity from either of the '1' side, the other side is a foreign key*(would be unique)*



If a binary relationship between two entity sets is 1-to-many,

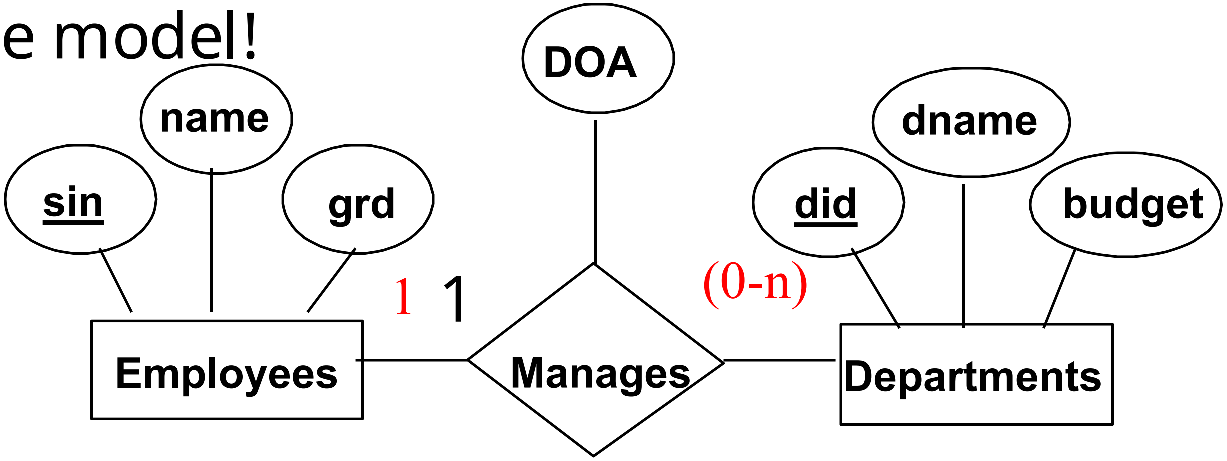
- the primary key of the relationship is from the 'm' side, the '1' side is the foreign key *(would be unique)*

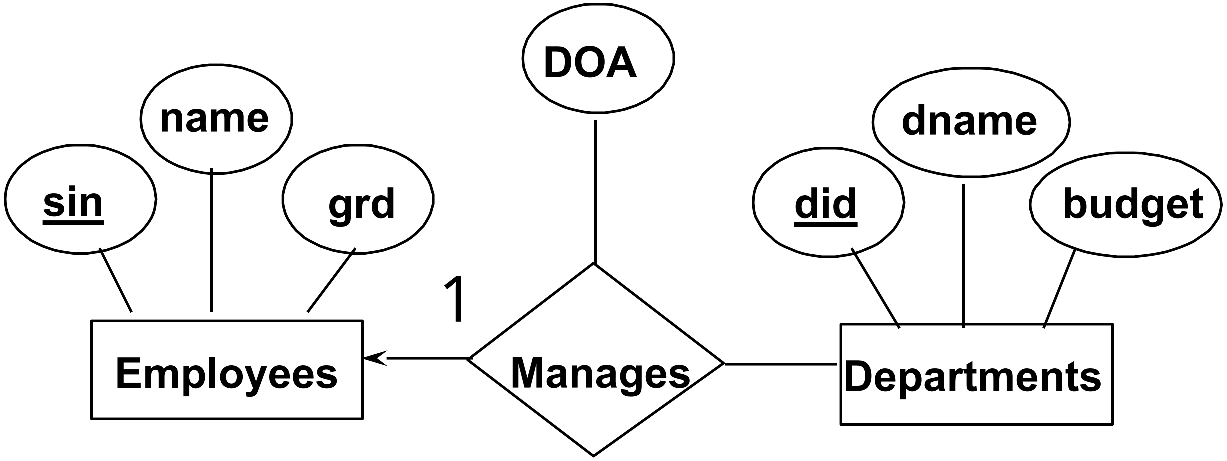


If a binary relationship between two entity sets is m-to-n,

- the primary key is composite, consisting of the primary key of the entities from each side of the relationship

Alternate methods of showing the same model!





- Map relationship to a table:
  - Note that did is the key now!
  - Separate tables for Employees and Departments.

```
CREATE TABLE  Manages
( sin  CHAR(9),
 did  INTEGER,
 DOA DATE,
 PRIMARY KEY  (did),
 FOREIGN KEY (sin) REFERENCES Employees,
 FOREIGN KEY (did) REFERENCES Departments)
```

- Since each department has a unique manager, we could instead combine Manages and Departments.

```
CREATE TABLE  DeptMgr
  (did  INTEGER,
  dname  CHAR(20),
  budget  REAL,
  sin  CHAR(9),
  DOA  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY (sin) REFERENCES Employees)
```

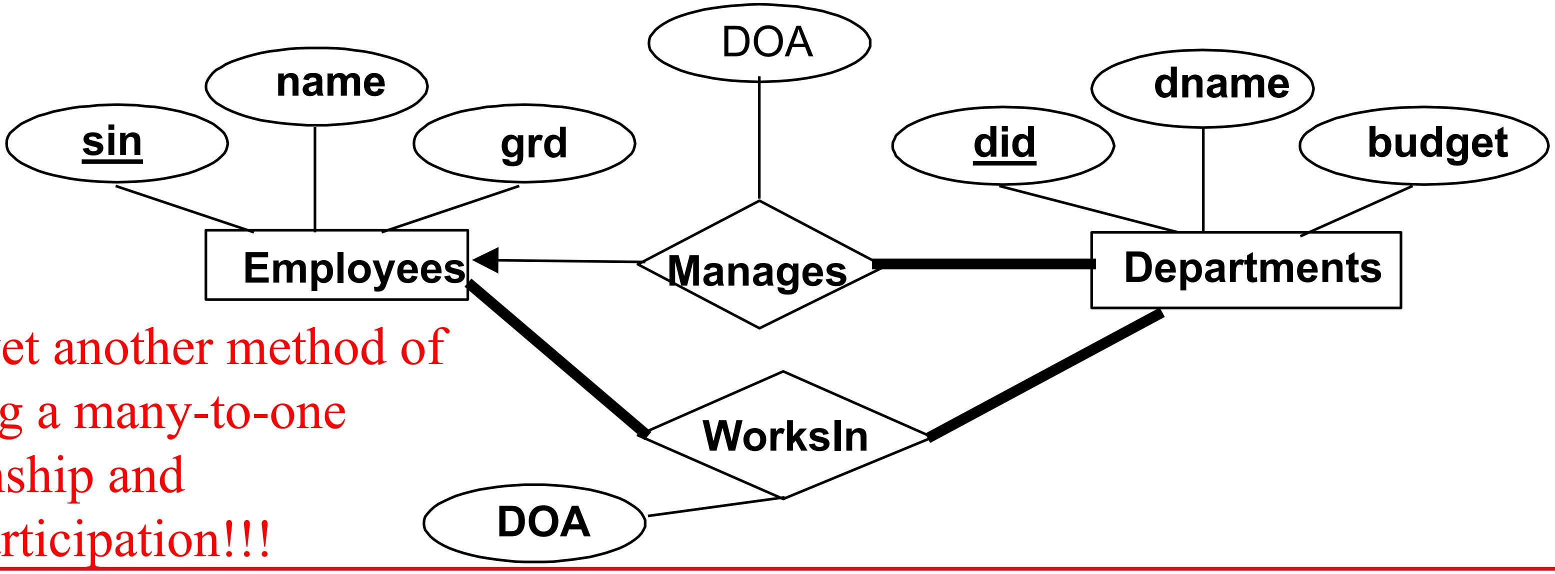
Null for Dept. w/o manager!

Participation Constraints

- Every department has a manager (a business rule) ⇒ *participation constraint*:
  - The participation of Departments in Manages is *total*

(all instances of Department must have a manager; participation of Employees is *partial i.e., not all employees are managers*).

Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *sin* value!)



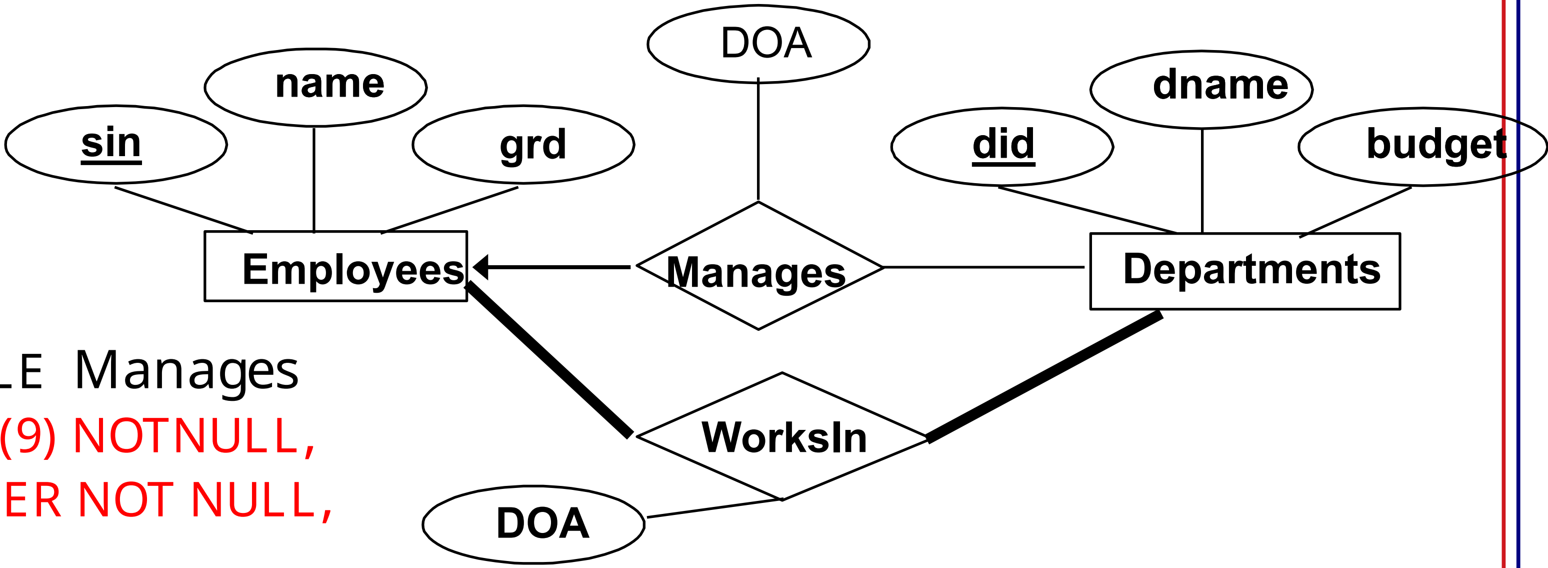
Note: yet another method of showing a many-to-one relationship and total participation!!!

Participation Constraints: SQL

- A participation constraints involving one entity set in a binary relationship, can be expressed as follows without resorting to CHECK constraints.

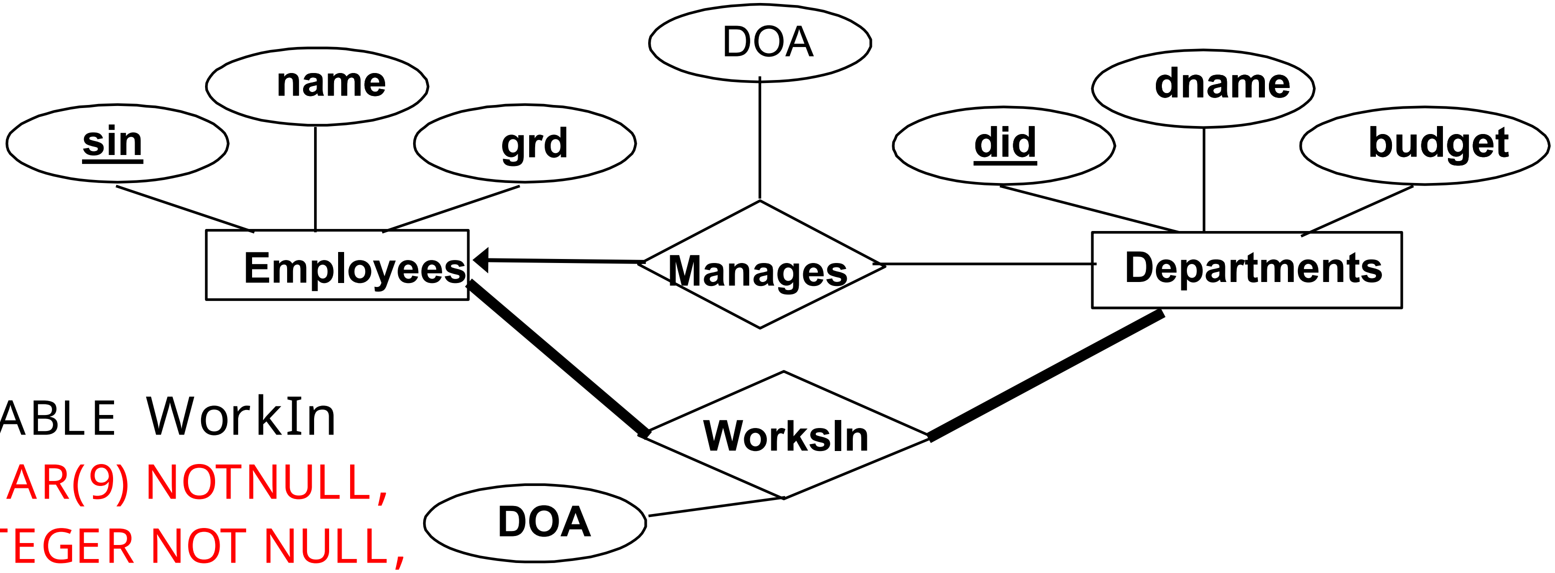
```
CREATE TABLE  DeptMgr
  (did  INTEGER,
  dname  CHAR(20),
  budget  REAL,
  sin  CHAR(9) NOT NULL,
  DOA  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY  (sin) REFERENCES Employees,
     ON DELETE NO ACTION)
```

Every department must have a manager!



```
CREATE TABLE  Manages
  ( sin  CHAR(9) NOTNULL,
   did  INTEGER NOT NULL,
   DOA DATE,
   PRIMARY KEY  (did),
   FOREIGN KEY (sin) REFERENCES Employees,
   FOREIGN KEY (did) REFERENCES Departments)
```

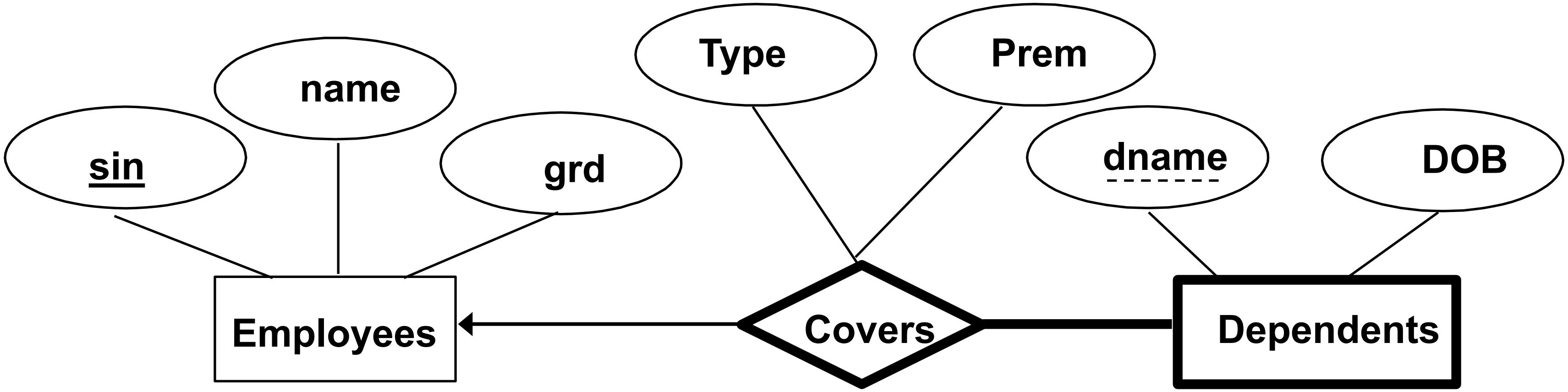
Here a department can exist without a manager.
We couldn't insert an occurrence of this relation
without having an occurrence of a department
and employee! Once inserted, does firing the manager
create problems?



```
CREATE TABLE  WorkIn
 ( sin  CHAR(9) NOTNULL,
  did  INTEGER NOT NULL,
  DOA DATE)
```

To ensure total participation of department in WorkIn, each did value must be in at least one tuple of WorkIn: enforced by assertion

- A *weak entity* can be identified uniquely only by considering the primary key of another *strong*-owner entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.

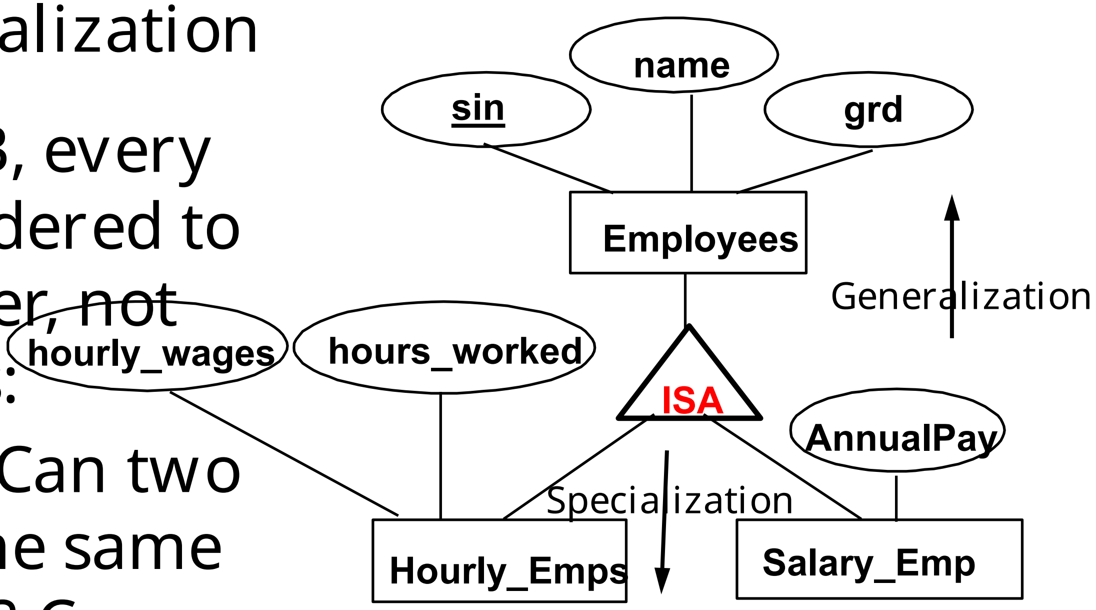


- Weak entity set and identifying relationship set are translated into a single relation.
  - Weak entity ⇒ total participation
  - When the owner entity is deleted, all owned weak entities must also be deleted.

```
CREATE TABLE  Covers
 ( dname  CHAR(20),
   DOB  DATE,
   Type INTEGER,
   Cost FLOAT,
   sin  CHAR(9) NOT NULL,
   PRIMARY KEY  (dname, sin),
   FOREIGN KEY  (sin) REFERENCES Employees,
      ON DELETE CASCADE)
```

Generalization, Specialization

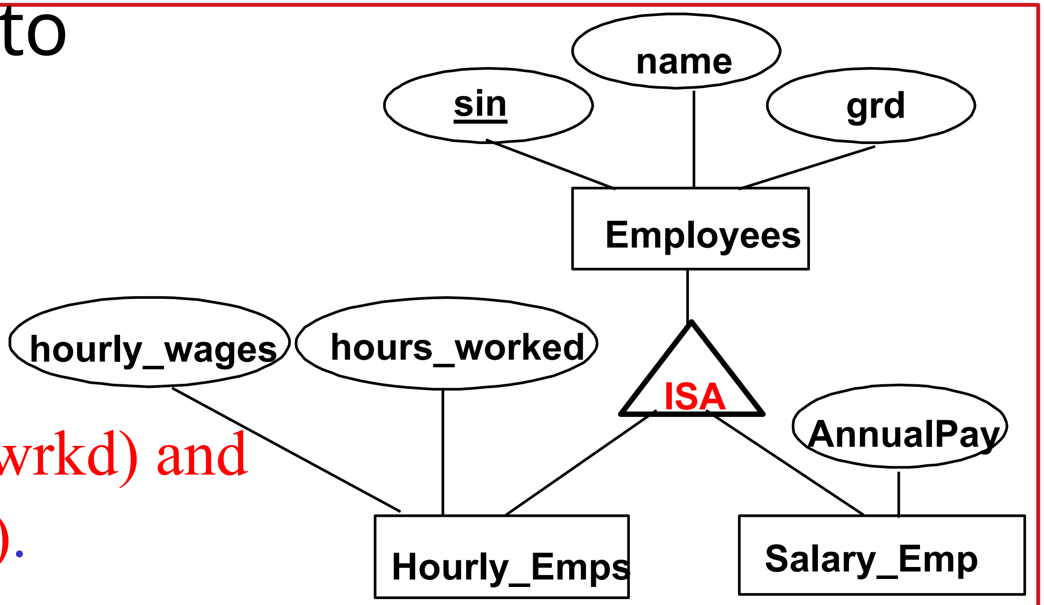
- If we declare A **ISA** B, every A entity is also considered to be a B entity. However, not implemented always:
- *Overlap constraints*: Can two subclasses contain the same instance of an entity? Can Carole be an Hourly_Emps as well as a Salary_Emps? (*Allowed/disallowed*)



Covering constraints: Does every Employees entity also have to be an Hourly_Emps or a Contract_Emps entity? *(Yes/no)*

- Reasons for using ISA relationship:
  - To add attributes specific to a subclass.
  - To identify subset of an entity set that participate in a relationship.

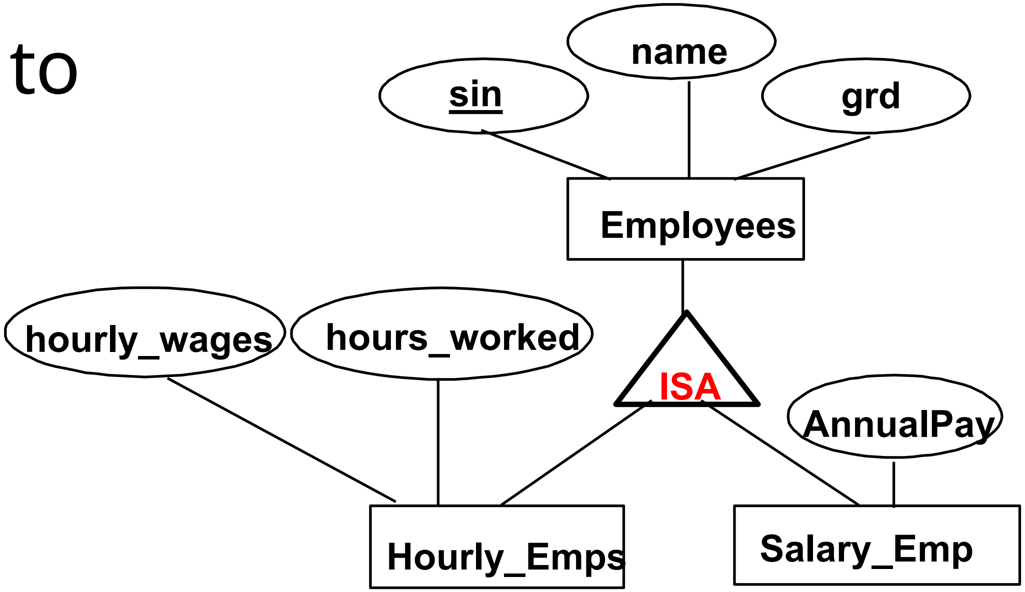
ISA relationship to Relations



Create 3 relations:
Employees(Sin, Name, Grd),
Hourly_Emps(Sin, Hwage, Hwrkd) and
Salary_Emps(Sin, AnnualPay).

- Every employee is recorded in Employees. For hourly employees, value for additional attribute are recorded in Hourly_Emps; if referenced Employees tuple is deleted, Hourly_Emps tuple must also be deleted.
- Queries involving all employees easy, those involving just Hourly_Emps require a join with Employee to get inherited attributes.

ISA relationship to Relations



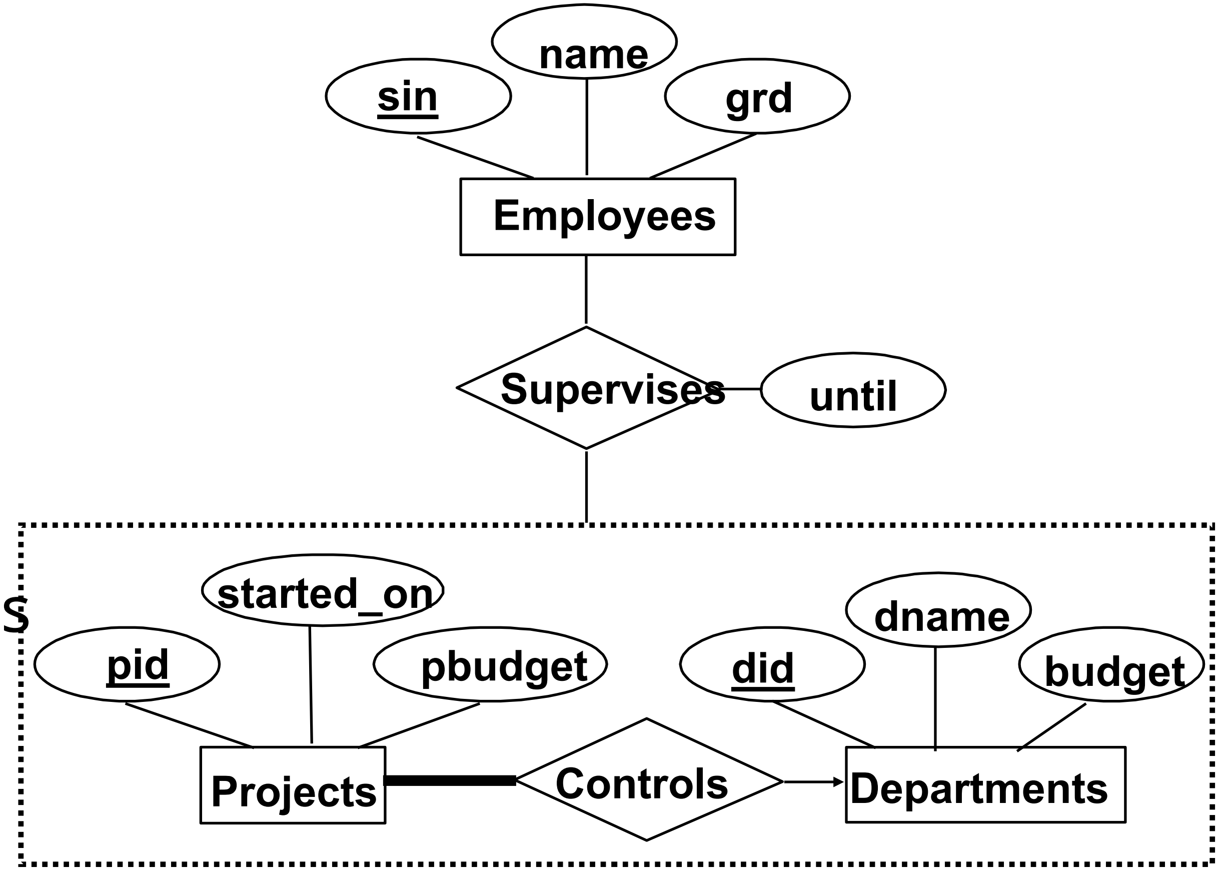
- Create two relations:
- Hourly_Emps(Sin, Name, Grd, HWrkd, Hwages) and Salary_Emps (Sin, Name, Grd, AnnualPay).

Each employee must be in one of these two subclasses.

All employees require accessing Two relations

Aggregation

- ***Aggregation***: models a relationship, involving entity sets and a *relationship* set, as an entity set. The aggregated entity participates in other relationships.
  - Supervise mapped to table like any other relationship set.

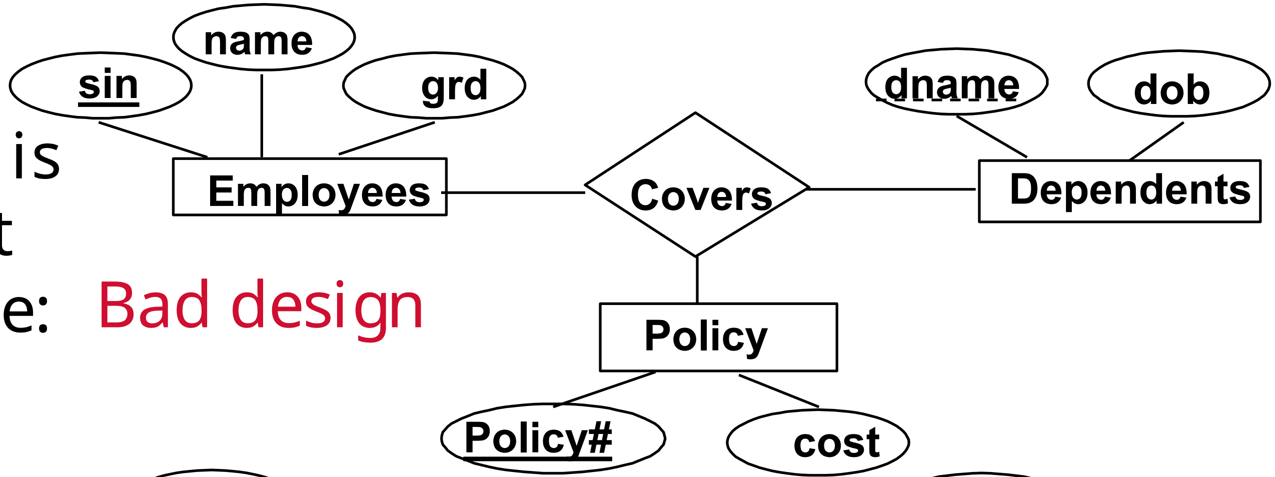


 Aggregation vs. ternary relationship:

- Supervises is a distinct relationship, with its attribute.

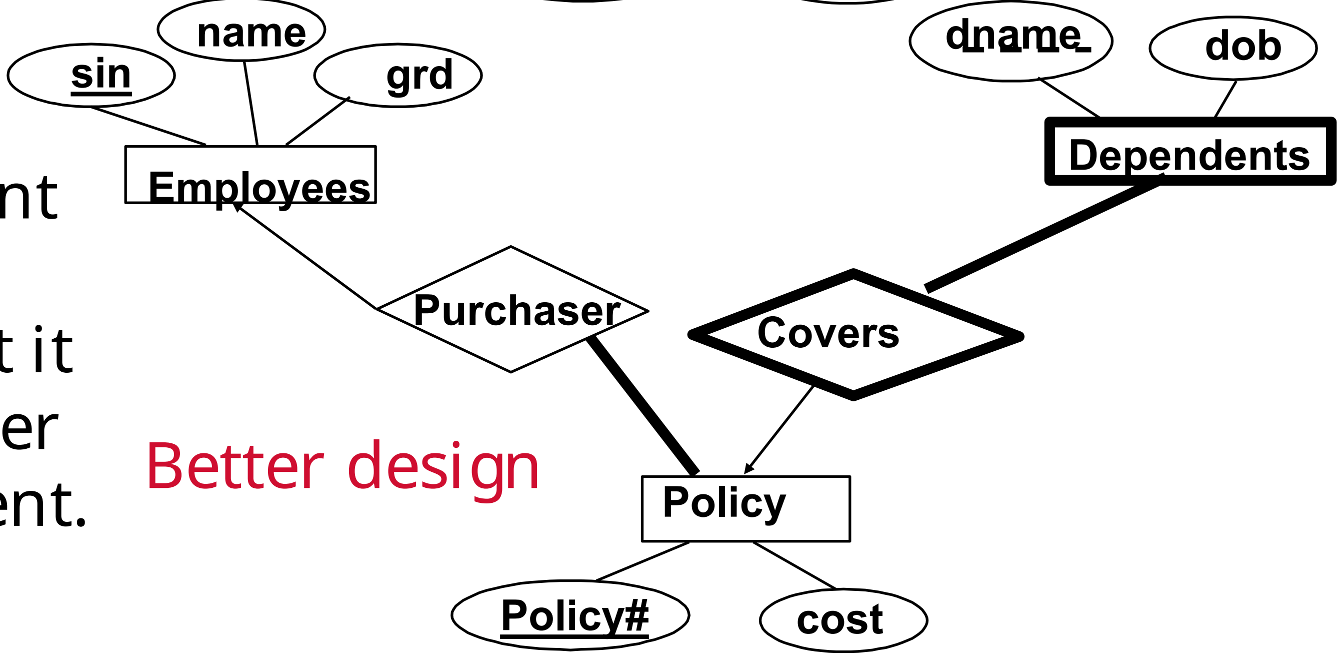
Binary vs. Ternary Relationships

- If each policy is owned by just ONE employee:



Bad design

- Key constraint on Policy requires that it can only cover one dependent.

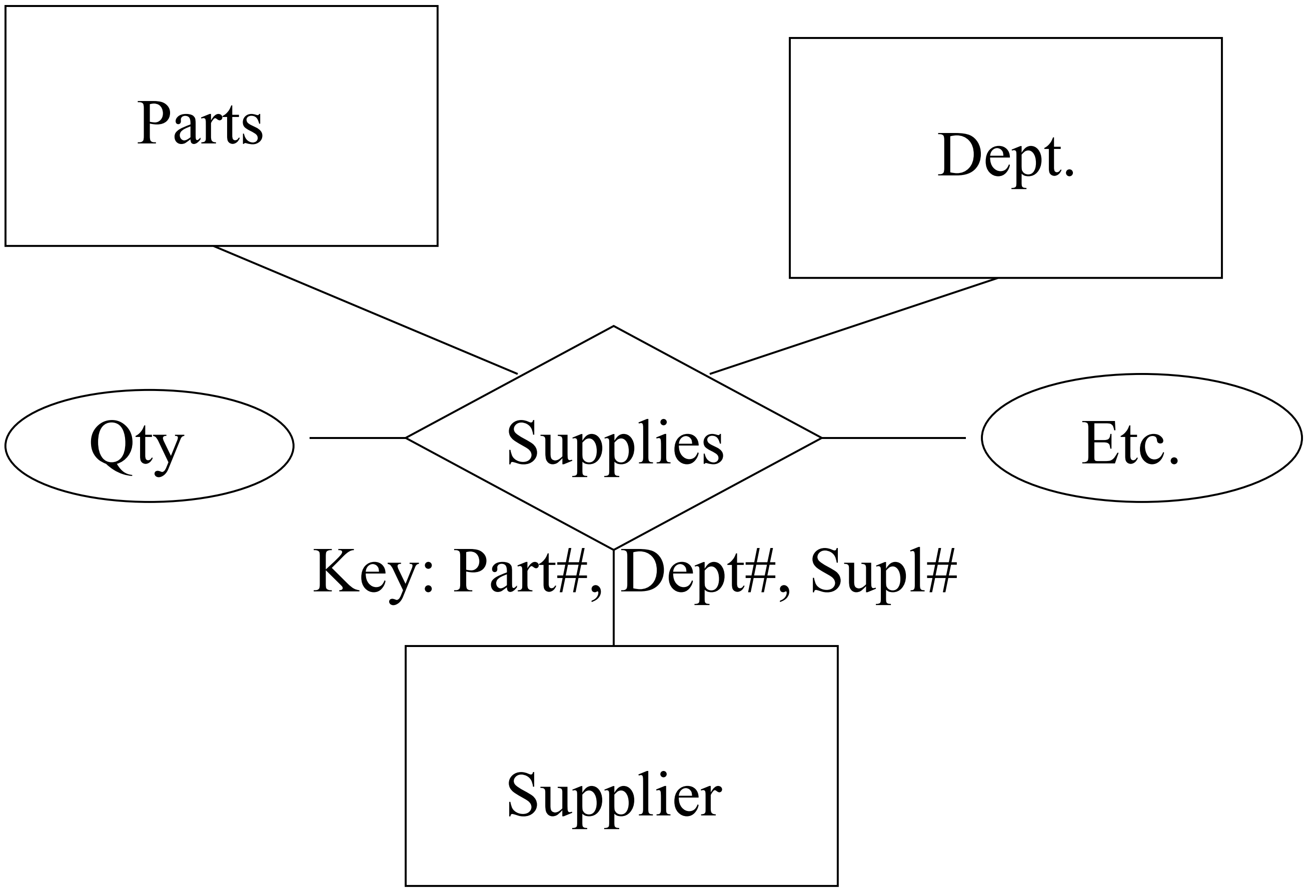


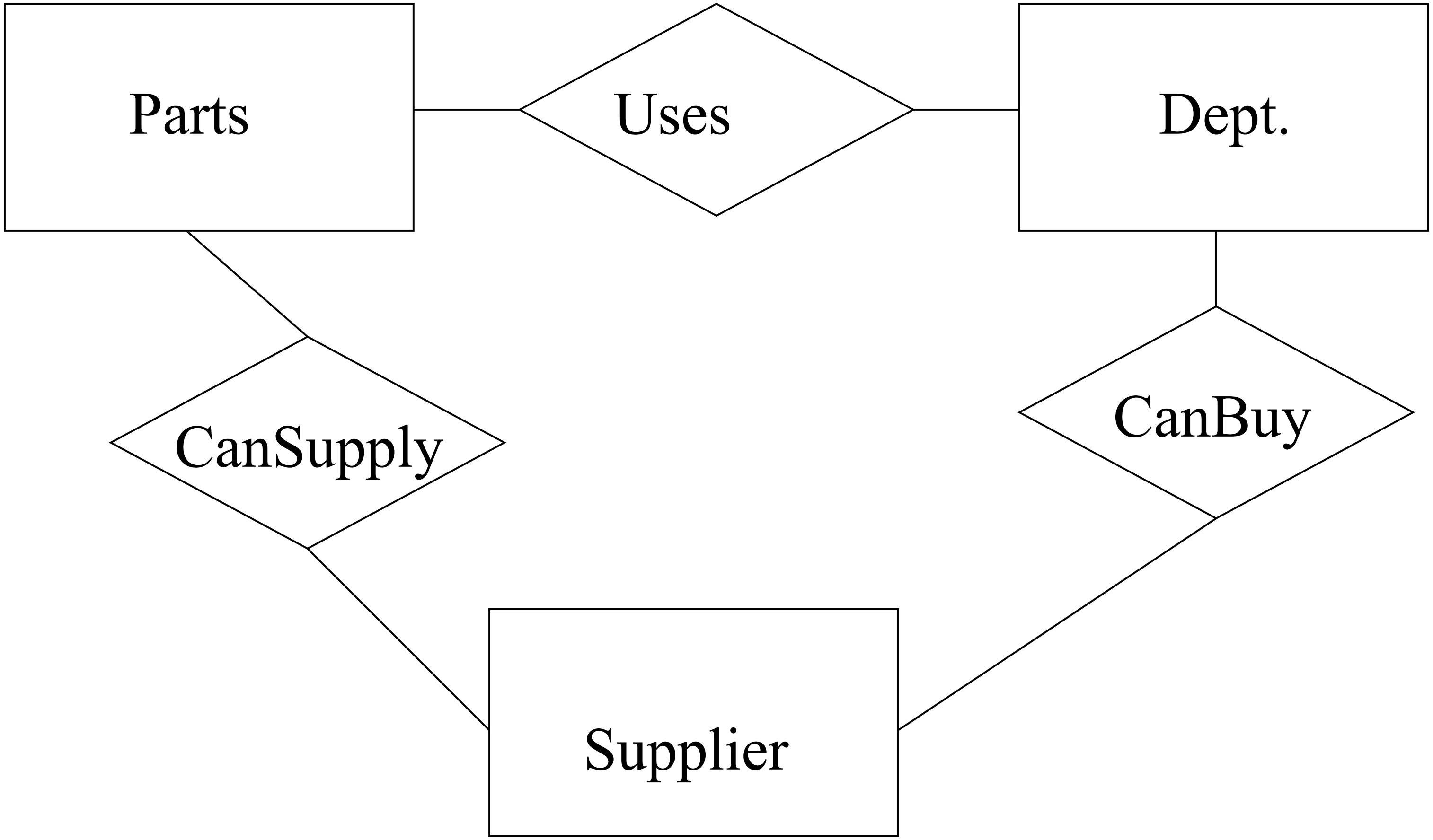
Better design

- The key constraints allow us to combine Purchaser with Policy and Covers with Dependents.
- Participation constraints lead to NOT NULL constraints.

```
CREATE TABLE  Policy (
   policy# INTEGER,
   cost  REAL,
   sin  CHAR(9)  NOT NULL,
   PRIMARY KEY (policy#).
   FOREIGN KEY (sin) REFERENCES
              Employees,
     ON DELETE CASCADE)

CREATE TABLE Dependents (
   dname  CHAR(20),
   dob  DATE,
   policy# INTEGER,
   PRIMARY KEY (dname, policy#).
   FOREIGN KEY (policy#)
             REFERENCES Policy,
     ON DELETE CASCADE)
```

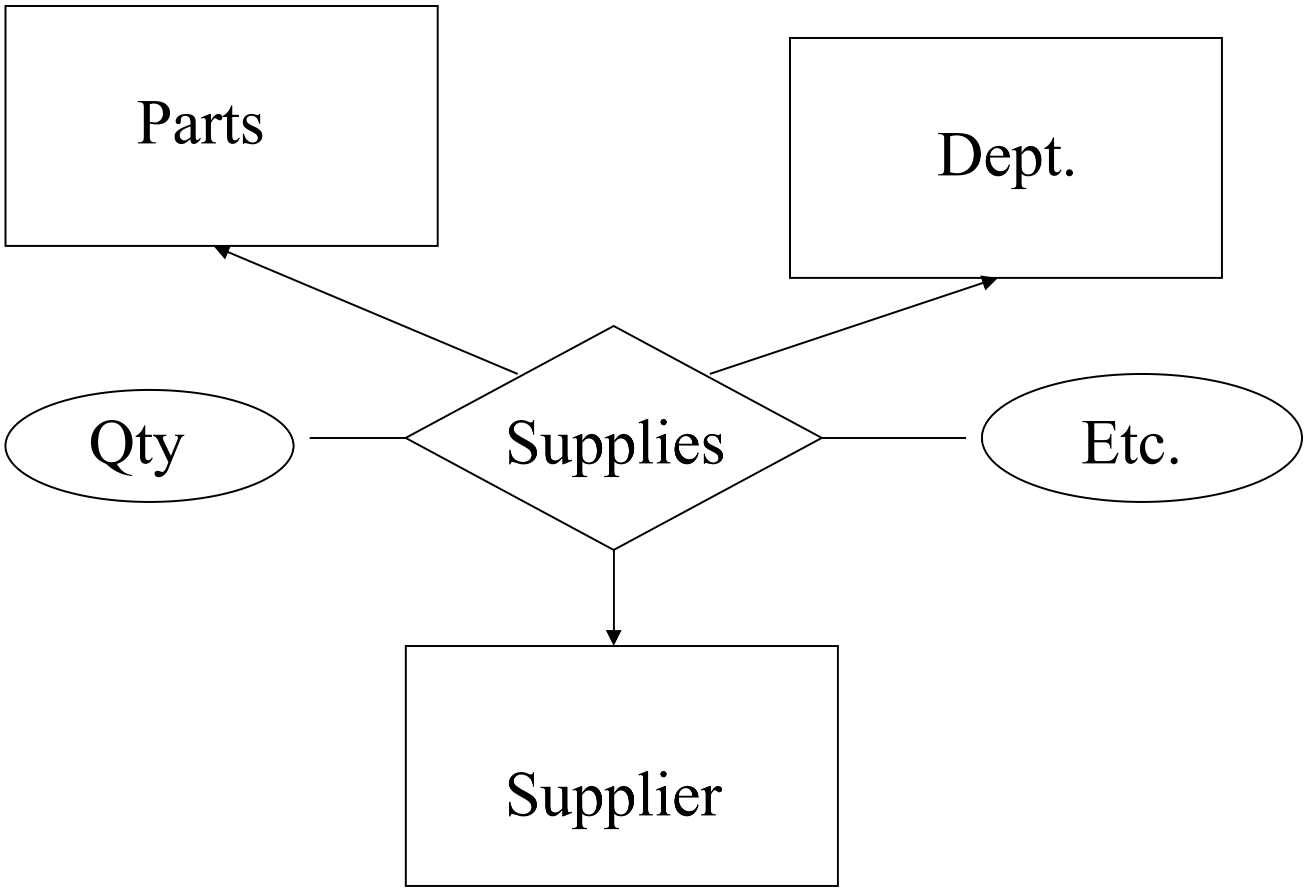
An example where a ternary relation is better than a number of binary relations is the following:

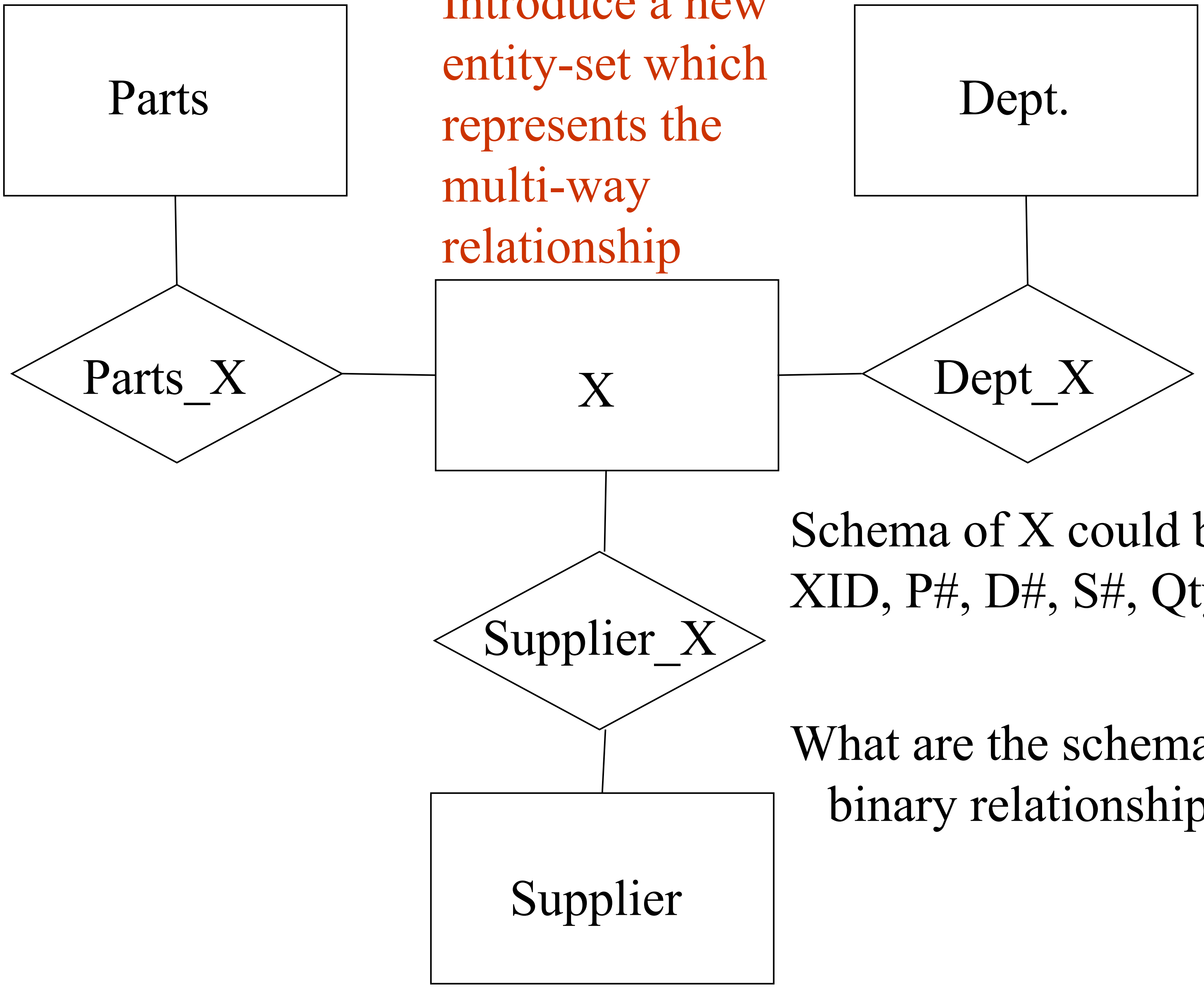



Parts
Uses
Dept.
CanSupply
CanBuy
Supplier

- Supplies relates entity sets Parts, Departments and Suppliers, and has descriptive attributes price, quantity etc.
- A number of binary relationships may not convey the semantics
- Supplier ``CanSupply'' Part,  Dept. ``Uses'' Part, and Dept. ``Can Buy'' from Supplier does not imply that Dept has a PO to buy Part from S.
  - How do we record the following: which part, quantity price?

A department can order only one part from a supplier?





Introduce a new entity-set which represents the multi-way relationship

Schema of X could be XID, P#, D#, S#, Qty, …

What are the schema of the binary relationship?

Constraints Beyond the ER Model

- Functional dependencies:
  - *A department can order only one part from a given supplier.*
    - Can’t express this in ternary Supplies relationship.
  - Normalization refines ER design by considering FDs.
- Inclusion dependencies:
  - Special case: Foreign keys (ER model can express these).
  - *e.g., At least 1 person must report to each manager.* (Set of *sin* values in Manages must be subset of *supervisor_ssn* values in Reports_To.)
- General constraints:
  - *e.g., Manager’s discretionary budget less than 10% of the combined budget of all departments he or she manages.*

Course Notes

Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

Databases – the generations

Bipin C. DESAI

Pl. see: https://users.encs.concordia.ca/~bcdesai/CopyForward.pdf

FIRST GENERATION

1950s –Refinement of storage media, magnetic tape, drums, disks

Early 1960s: Disk access method based on
Index Sequential Access Method(ISAM)

Mid 1960s:Emergence - Information Management System(IMS)-IBM developed in 1966 along with NASA(Rockwell and Caterpillar) to support the Apollo/Saturn V program

Current version is IMS 15.4 and runs on IBM z platform

It is still being marketed, used in banking etc.
promises > $250*10^9$ transactions per day

1959 : CODASYL(**Conf./Committee on Data Systems Languages)**
later to become **Database Task Group (**DBTG),
DBTG developed the network model and its implementation
Integrated Data Store (IDS),
Integrated Database Management System (IDMS)
both still marketed and supported.

SECOND GENERATION

1970 Codd’s paper about relations

1973/1974 Ingres(UC Berkley, M. Stonebraker, E.Wong)

System R(IBM), Berkley/DB (Sleepy Cat Software, Oracle)

QUEL, SEQUEL(Ingres) and SQL(System R)

1978 Oracle

1981 Informix (IBM)

1984 System R(IBM)

1987 Postgres

1993 mSQL (mini SQL by D. Hughes)

mSQL used in the development of early dynamic Web applications including CrsMgr and ConfSys

1995 MySQL - bought by Sun in 2008 price- $1billion

– Sun was taken over by Oracle

2009 Mariadb – a fork of MySQL

THIRD GENERATION

2004 MapReduce paradigm shift to lower level!
Map(distribute tasks to nodes to filter local data) and then
Reduce(process result in parallel)
2005 Hadoop (Apache)
2008 Cassandra, Hbase,
2009 MongoDB

Simple SQLPlus & SQL

Bipin C. DESAI

Getting & Installing {Apache, Oracle, PHP} or, XAMPP

Consult:

http://www.oracle.com/technology/tech/php/htdocs/inst_php_apache_windows.html
or whatever is the currrent URL
For Oracle you need to register with OTN

MySQL/Mariadb

https://www.apachefriends.org/download.php

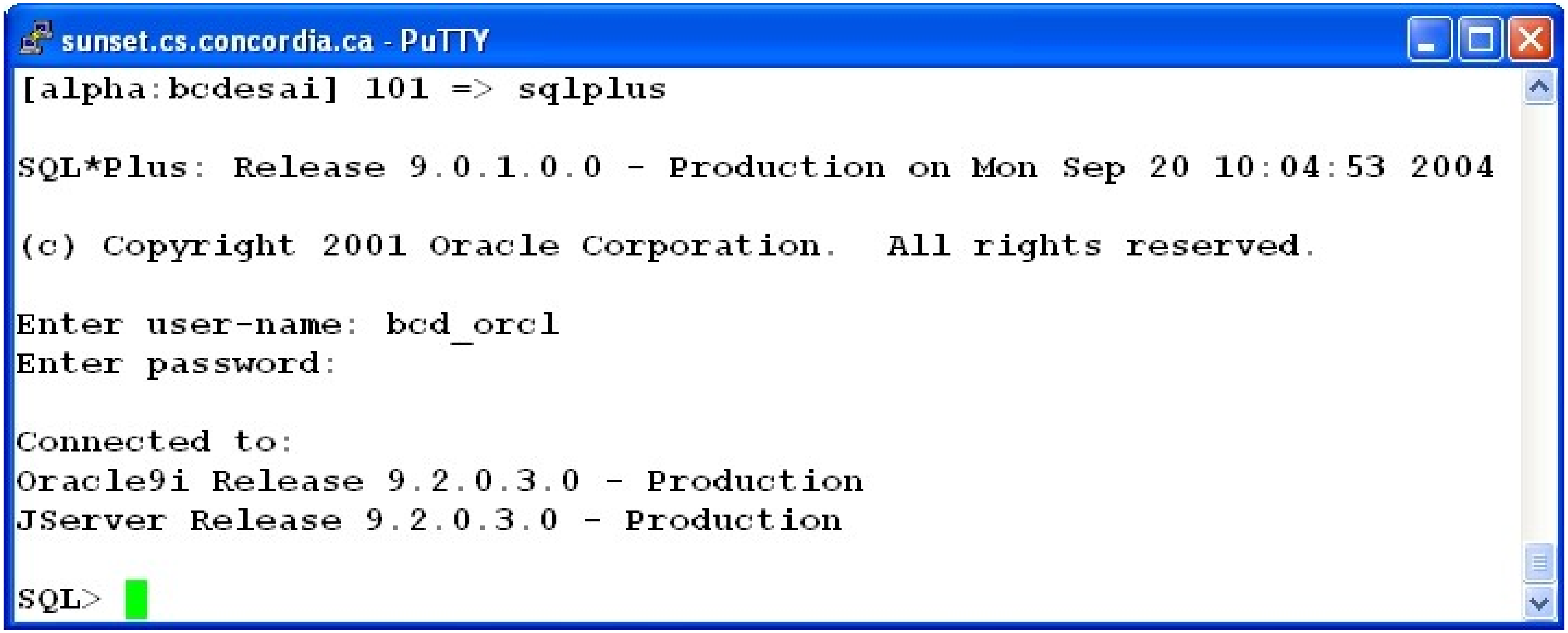
The projects are to be demonstrated on one of the systems in our labs.
So if you develop the projects on your own systems, make sure you could:

- Upload all the code to CrsMgr
- Have it run on one of AITS systems which has one of the above configurations
- It works as specified

These notes uses Oracle, MySQL, MariaDB

Connecting to SQLPlus

SQLPlus is a “user friendly interface” to ORACLE SQL to be used interactively.
You need Oracle USERID/PASSWORD and appropriate permission to a Oracle DB.
May connect remotely using a secure shell (e.g., Putty)



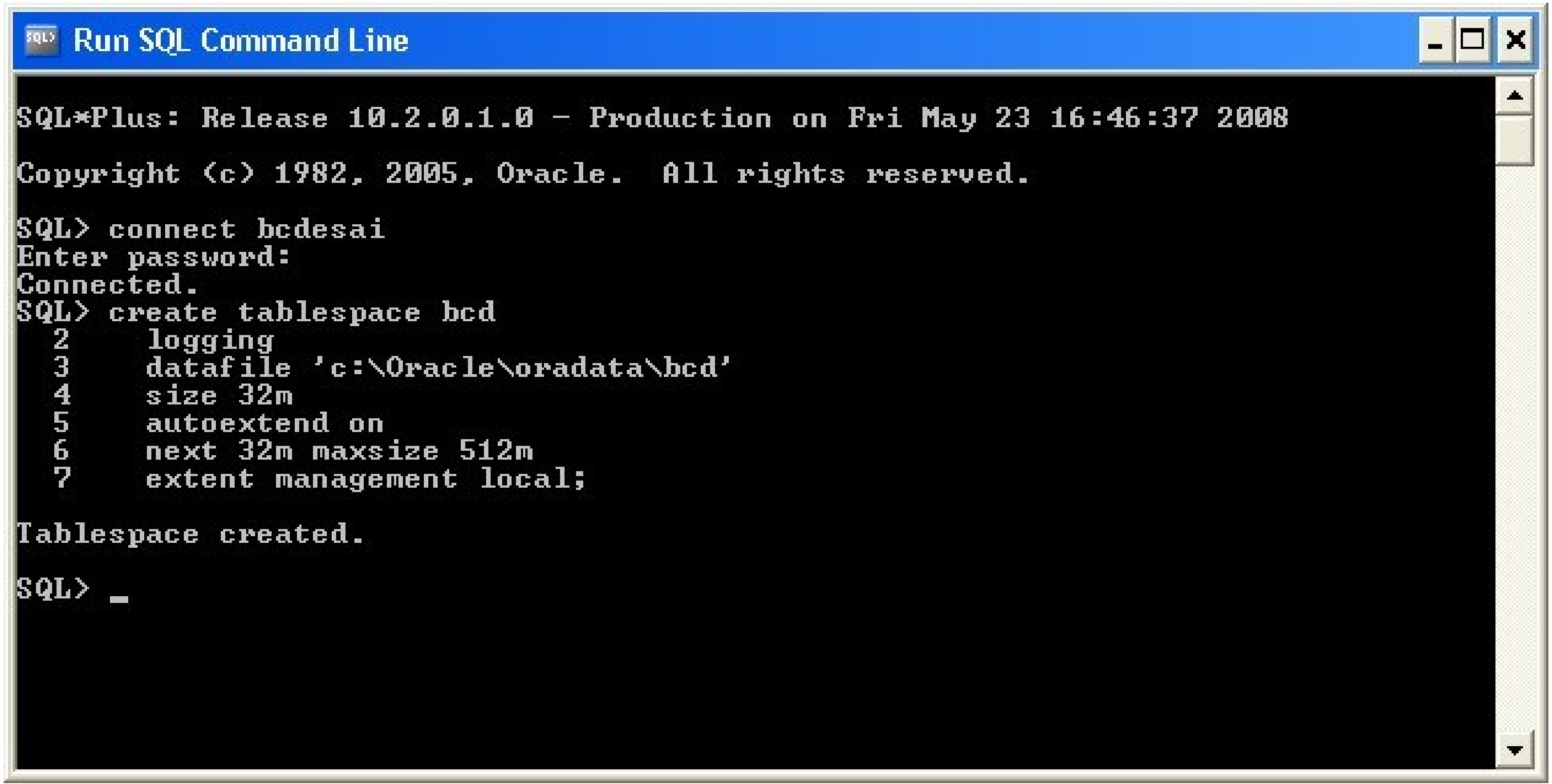
```
[alpha:bcdesai] 101 => sqlplus

SQL*Plus: Release 9.0.1.0.0 - Production on Mon Sep 20 10:04:53 2004

(c) Copyright 2001 Oracle Corporation.  All rights reserved.

Enter user-name: bcd_orcl
Enter password:

Connected to:
Oracle9i Release 9.2.0.3.0 - Production
JServer Release 9.2.0.3.0 - Production

SQL>
```

Download and install Oracle (the version changes over time)

Typically - start database (unless it has been installed as service which starts on boot)

From Start select RunSQL command line

Connect to oracle:



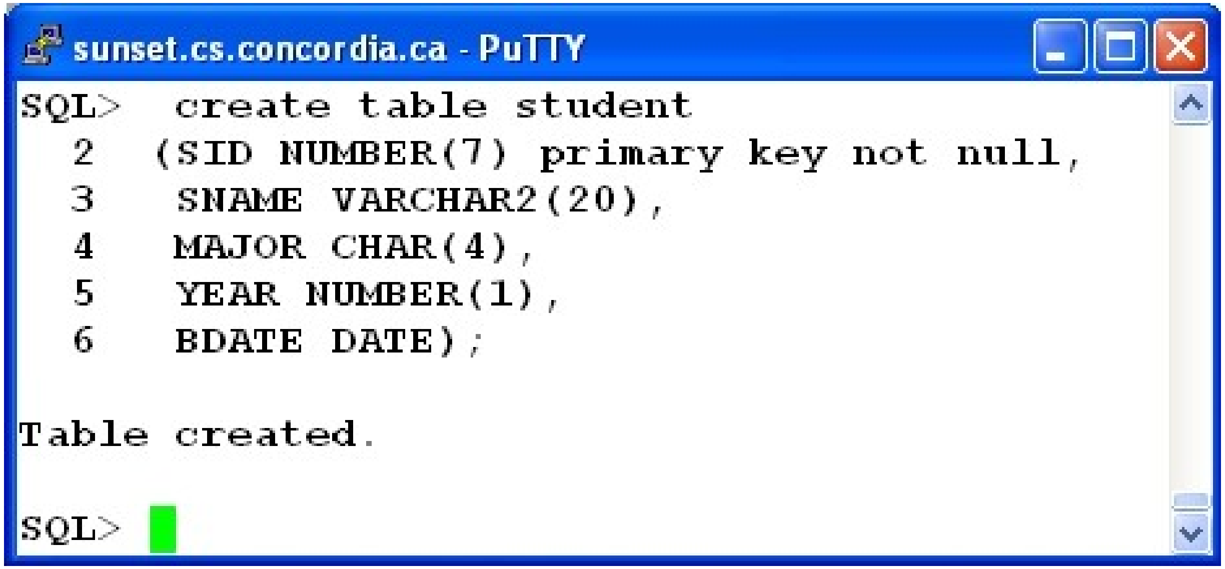
```
SQL*Plus: Release 10.2.0.1.0 - Production on Fri May 23 16:46:37 2008

Copyright (c) 1982, 2005, Oracle.  All rights reserved.

SQL> connect bcdesai
Enter password:
Connected.
SQL> create tablespace bcd
  2    logging
  3    datafile 'c:\Oracle\oradata\bcd'
  4    size 32m
  5    autoextend on
  6    next 32m maxsize 512m
  7    extent management local;

Tablespace created.

SQL> _
```

Bipin C. DESAI

```
create table student
(SID NUMBER(7) primary key not null,
 SNAME VARCHAR2(20),
 MAJOR CHAR(4),
 YEAR NUMBER(1),
 BDATE DATE)
 tablespace bcd pctfree 2;
```

To execute a text file containing sql statements interactively from the sql prompt use @ followed by the full path to file

sql>@student.sql



```
SQL>  create table student
  2  (SID NUMBER(7) primary key not null,
  3   SNAME VARCHAR2(20),
  4   MAJOR CHAR(4),
  5   YEAR NUMBER(1),
  6   BDATE DATE);

Table created.

SQL>
```

Connecting to MySQL/MariaDB

MySQl/Mariadb has a simpler text based interface
used for connecting to the database
running locally or on a server accessed using a terminal emulator
Putty is one used in WinX

Again the DB server must be running and one needs
a user ID and password for the database to be used

```
shell> mysql –u username –p password
```

If the ID/PW are correct, one gets the prompt from the database

```
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 96773
Server version: 10.3.17-MariaDB MariaDB Server

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and
others.

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

MariaDB [(none)]> connect test;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Connection id:    29348
Current database: test
```

```
mysql> create table student
(SID DECIMAL(7) primary key not null,
 SNAME VARCHAR (20),
 MAJOR CHAR(4),
 YEAR DEC(1),
 BDATE DATE);
```

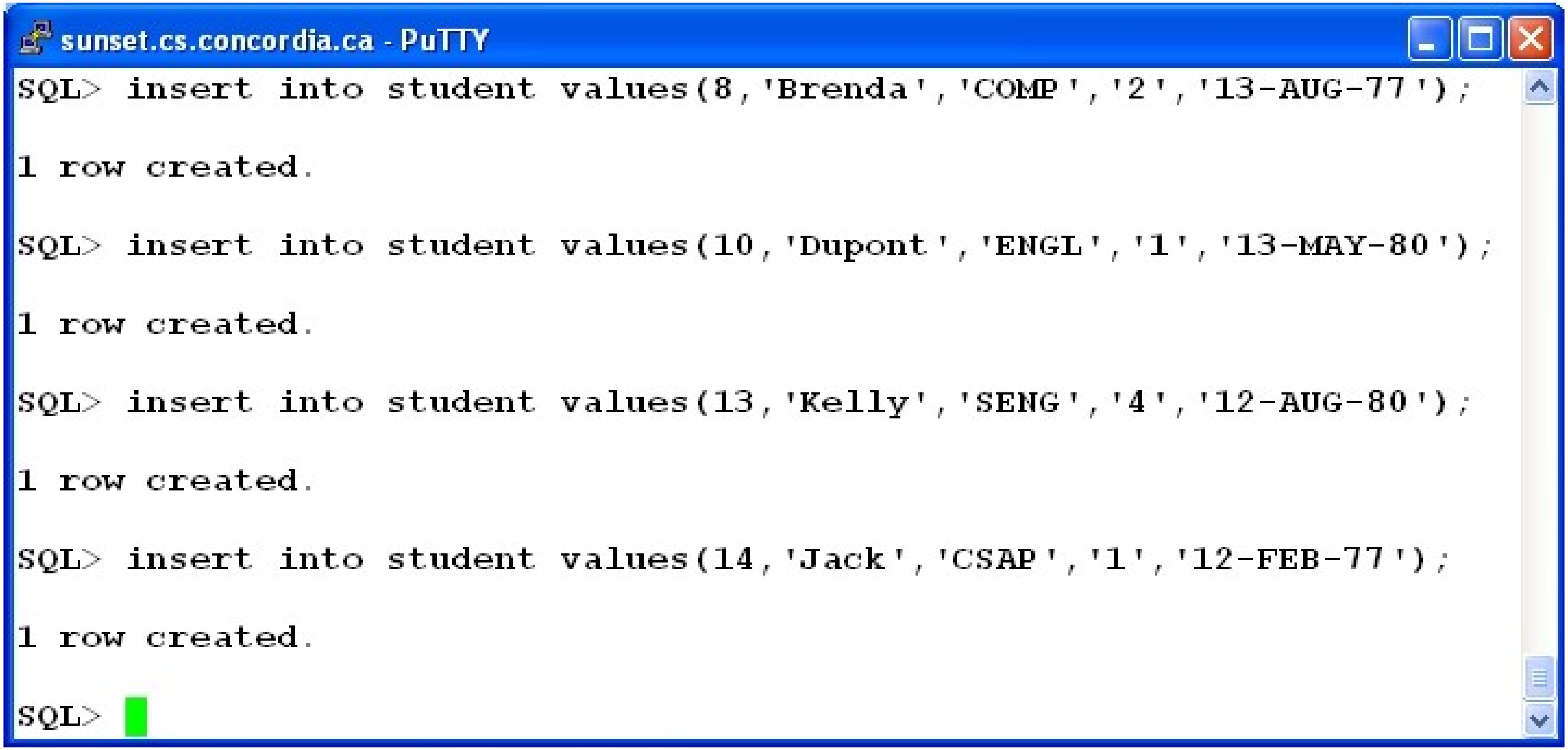
To execute a text file containing sql statements interactively from the sql prompt use @ followed by the full path to file

sql>@student.sql

in MySQL use “source student.sql”

```
mysql> desc student;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| SID   | decimal(7,0) | NO   | PRI | NULL    |       |
| SNAME | varchar(20)  | YES  |     | NULL    |       |
| MAJOR | char(4)      | YES  |     | NULL    |       |
| YEAR  | decimal(1,0) | YES  |     | NULL    |       |
| BDATE | date         | YES  |     | NULL    |       |
+-------+--------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

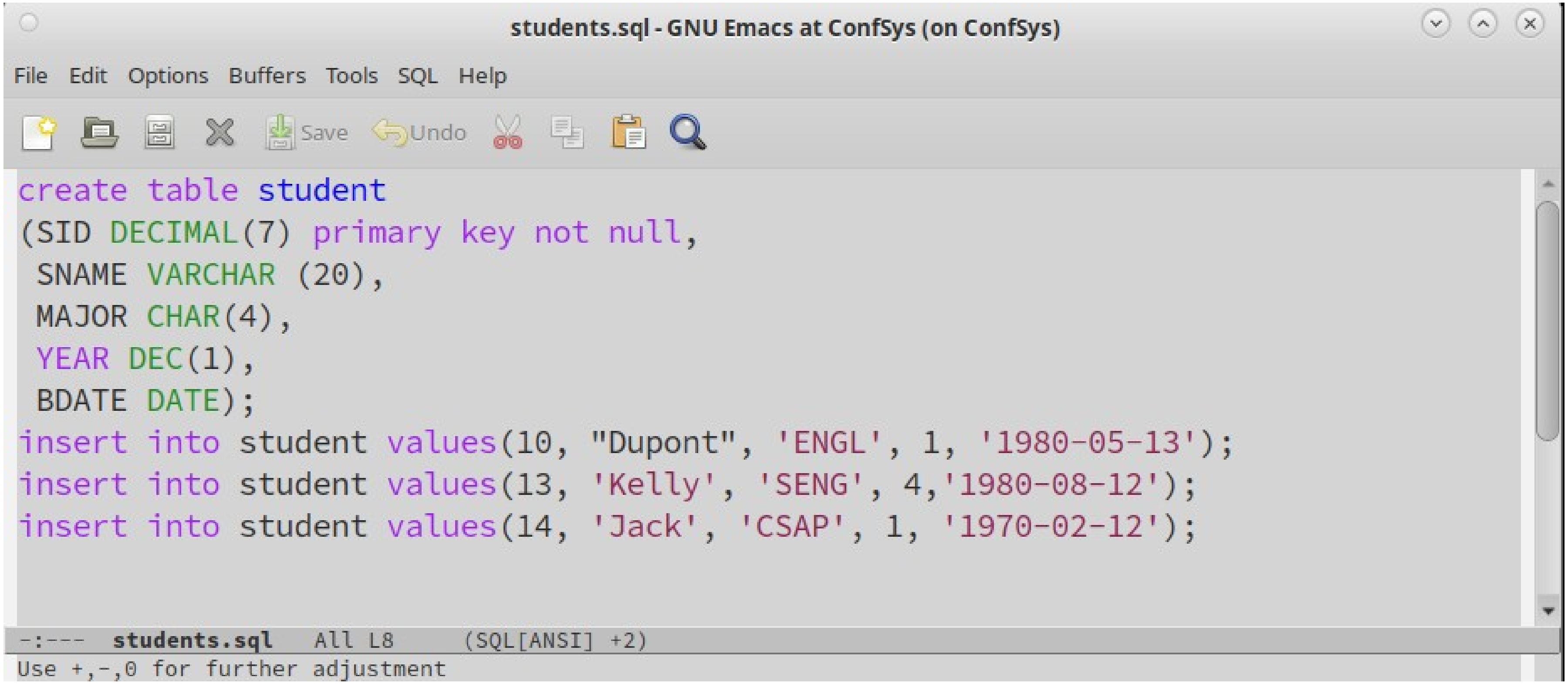
Inserting Data in a table – table must exist!



```
SQL> insert into student values(8,'Brenda','COMP','2','13-AUG-77');

1 row created.

SQL> insert into student values(10,'Dupont','ENGL','1','13-MAY-80');

1 row created.

SQL> insert into student values(13,'Kelly','SENG','4','12-AUG-80');

1 row created.

SQL> insert into student values(14,'Jack','CSAP','1','12-FEB-77');

1 row created.

SQL>
```

Date format in MySQL is yyyy-mm-dd;
Value order as in schema for the table
MariaDB [test]> insert into student values
(8, 'Brenda', 'COMP', 2, '1977-8-13');

```
MariaDB [test]> \! tcsh    -- escape to interative shell (tcsh)
101 => emacs -nw students.sql
 104 => more students.sql
insert into student values(10, "Dupont", 'ENGL', 1, '1980-05-13');
insert into student values(13, 'Kelly', 'SENG', 4,'1980-08-12');
insert into student values(14, 'Jack', 'CSAP', 1, '1970-02-12');
 105 => exit
exit
MariaDB [test]>system cat students.sql;
create table student
(SID DECIMAL(7) primary key not null,
 SNAME VARCHAR (20),
 MAJOR CHAR(4),
 YEAR DEC(1),
 BDATE DATE);
insert into student values(10, "Dupont", 'ENGL', 1, '1980-05-13');
insert into student values(13, 'Kelly', 'SENG', 4,'1980-08-12');
insert into student values(14, 'Jack', 'CSAP', 1, '1970-02-12');
MariaDB [test]>
```



students.sql - GNU Emacs at ConfSys (on ConfSys)
File Edit Options Buffers Tools SQL Help
Save
Undo

```
create table student
(SID DECIMAL(7) primary key not null,
 SNAME VARCHAR (20),
 MAJOR CHAR(4),
 YEAR DEC(1),
 BDATE DATE);
insert into student values(10, "Dupont", 'ENGL', 1, '1980-05-13');
insert into student values(13, 'Kelly', 'SENG', 4,'1980-08-12');
insert into student values(14, 'Jack', 'CSAP', 1, '1970-02-12');
```

```
-:---  students.sql   All L8     (SQL[ANSI] +2)
Use +,-,0 for further adjustment
```

```
MariaDB [test]>
MariaDB [test]> source students.sql;
Query OK, 1 row affected (0.028 sec)
Query OK, 1 row affected (0.050 sec)
Query OK, 1 row affected (0.050 sec)

MariaDB [test]> select * from student;
+-----+--------+-------+------+------------+
| SID | SNAME  | MAJOR | YEAR | BDATE      |
+-----+--------+-------+------+------------+
|  10 | Dupont | ENGL  |    1 | 1980-05-13 |
|  13 | Kelly  | SENG  |    4 | 1980-08-12 |
|  14 | Jack   | CSAP  |    1 | 1970-02-12 |
+-----+--------+-------+------+------------+
3 rows in set (0.001 sec)

MariaDB [test]>
```

Find all students (ORACLE)

SQL> select * from student;

```
      SID  SNAME                 MAJO       YEAR BDATE
---------- -------------------- ---- ---------- ---------
         8 Brenda               COMP          2 13-AUG-77
        10 Dupont               ENGL          1 13-MAY-80
        13 Kelly                SENG          4 12-AUG-80
        14 Jack                 CSAP          1 12-FEB-77
```

```
SQL>column major format a5
SQL>column sid format 9,9
SQL>column sname format a12
SQL>column major format a5
SQL>column year format 999
SQL>column bdate format a12
```

format not available in MySQL

```
 SID SNAME        MAJOR YEAR BDATE
---- ---------- ----- ---- ----------
   8 Brenda       COMP     2 13-AUG-77
 1,0 Dupont       ENGL     1 13-MAY-80
 1,3 Kelly        SENG     4 12-AUG-80
 1,4 Jack         CSAP     1 12-FEB-77
```

```
MariaDB [test]> select * from student;
+-----+--------+-------+------+------------+
| sid | sname  | major | year | bdate      |
+-----+--------+-------+------+------------+
|   8 | Brenda | COMP  |    2 | 1997-08-13 |
|  10 | Dupont | ENGL  |    1 | 1980-05-13 |
|  13 | Kelly  | SENG  |    4 | 1980-08-12 |
|  14 | Jack   | CSAP  |    1 | 1970-02-12 |
+-----+--------+-------+------+------------+
4 rows in set (0.001 sec)
```

```
select s.sname
from student s
where to_date(s.bdate) like '%13%';
```

```
SNAME
------------
Brenda
Dupont
```

```
select s.sname
from student s
where s.bdate like '%13%';
```

```
+---------+
| sname   |
+---------+
| Brenda  |
| Dupont  |
+---------+
2 rows in set (0.000 sec)
```

SQL script: date.sql

Find students born in August

```
select s.sname
from student s
where to_date(s.bdate) like '%AUG%';
```

```
SNAME
------------
Brenda
Kelly
```

```
select s.sname
from student s
where s.bdate like '%-08-%';
```

```
+---------+
| sname   |
+---------+
| Brenda  |
| Kelly   |
+---------+
2 rows in set(0.000 sec)
```

SQL script: month.sql

Find student born in 1977

```
select s.sname
from student s
where to_date(s.bdate) like '%77%';
```

```
SNAME
------------
Brenda
Jack
```

SQL script: year.sql

```
select s.sname  from student s
where s.bdate like '%80-%';
```

```
+---------+
| sname   |
+---------+
| Dupont  |
| Kelly   |
+---------+
2 rows in set (0.001 sec)
```

```
create table dept
(DEPT CHAR(20) not null,
 CODE CHAR(4) primary key not null);

insert into dept values('Computer Science', 'COMP');
insert into dept values('Decision Science', 'DISC');

create table deptmajor
(CODE CHAR(4),
 MAJOR CHAR(20),
primary key (CODE, MAJOR))

insert into deptmajor values('COMP', 'COTH');
insert into deptmajor values('COMP', 'SENG');
insert into deptmajor values('COMP', 'CSAP');
insert into deptmajor values('DISC', 'OPRS');
```

```
create table course
(CNAME CHAR(20),
 CNUMBER CHAR(8) primary key NOT NULL,
 CREDITS NUMBER(2),
 ODEPT CHAR(4),
 foreign key (ODEPT) references dept(code)
 on delete cascade)

insert into course values('C++','COMP248',3,'COMP');
insert into course values('DATA STRUCTURES ','COMP352',3,
 'COMP');
insert into course values('OPERATING SYSTEMS','COMP346',4
,'COMP');
insert into course values('DATABASE','COMP353',4,'COMP');
insert into course values('Operation Research','DISC253',4,'DISC');
```

```
create table crs_section
(SECID NUMBER(6) primary key NOT NULL,
 COURSE_NUM CHAR(8),
 SECTION CHAR(2),
 SEMESTER CHAR(4),
 YEAR CHAR(4),
 SCHEDULE CHAR(10),
 ROOM CHAR(7));

insert into crs_section values
(85,'COMP352','A','FALL', '1998','TH16001715','H123');
insert into crs_section values
(90,'COMP353','B','FALL','1999','MW08451000','H631');
insert into crs_section values
(95,'DISC253','B','FALL','1999','MW10151130','H631');
```

```
create table prereq
(COURSE_Number CHAR(8),
 PREREQ CHAR(8),   primary key (course_number, prereq));
insert into prereq values('COMP353','COMP352');

insert into prereq values('COMP353','COMP346');
insert into prereq values('COMP352','COMP248');

create table enrollment
(STUDENT_NUMBER NUMBER(3) not null,
 SECTION_ID NUMBER(6) not null,  GRADE CHAR(1),
 primary key(student_number, section_id));
insert into enrollment values(8,85,null);
insert into enrollment values(10,90,null);
insert into enrollment values(8,90,null);
insert into enrollment values(14,90,null);
insert into enrollment values(14,95,null);
```

Find details of studs. taking a course offered by the “DISC” dept.

```
select s.SID, s.SNAME, s.MAJOR, s.YEAR, s.BDATE
from student s, dept d, course c, crs_section r, enrolment e
where  c.ODEPT=d.CODE and
       r.COURSE_NUM=c.CNUMBER and
       r.SECID=e.SECTION_ID and
       e.STUDENT_NUMBER = s.SID and
       d.CODE= 'DISC';
```

```
SID SNAME            MAJOR YEAR BDATE
---- ------------ ----- ---- ------------
 1,4 Jack             CSAP     1 12-FEB-77

SQL script: ex-select3.sql
```

Find student who are registered in a course offered by their majoring dept.

```
select * from student
where student.sid in
(select s.sid from student s, dept d, course c, crs_section r, enrollment e
where
c.ODEPT=d.CODE and             -- c Offering Dept same as the d dept
s.MAJOR=c.ODEPT and            -- s major Dept same as the c.ODEPT
r.COURSE_NUM=c.CNUMBER and   -- the section is for the course c
r.SECID=e.SECTION_ID and       -- r course section same as e section
e.STUDENT_NUMBER = s.SID);
```

```
 SID SNAME       MAJOR   YEAR BDATE
----   ------------     ----------  -------   -----------
   8     Brenda          COMP     2       13-AUG-80
```

Find students who are currently registered.

sql > @ex-select1.sql

```
select * from student
where student.sid in
        (select s.sid
        from student s, dept d, course c, crs_section r, enrolment e
        where  c.ODEPT=d.CODE and
r.COURSE_NUM=c.CNUMBER and
r.SECID=e.SECTION_ID and        e.STUDENT_NUMBER =
s.SID);
```

```
SID  SNAME          MAJOR  YEAR  BDATE
---- ------------ ----- ---- ------------
   8 Brenda        COMP     2 13-AUG-80
 1,0 Dupont        ENGL     1 13-MAY-80
 1,4 Jack          CSAP     1 12-FEB-77
```

```
select s.SID, s.SNAME, s.MAJOR, s.YEAR, s.BDATE
from student s, dept d, course c, crs_section  r, enrolment e
where  c.ODEPT=d.CODE and
       r.COURSE_NUM=c.CNUMBER and
       r.SECID=e.SECTION_ID and
       e.STUDENT_NUMBER = s.SID and
       d.CODE= 'COMP';
```

```
SQL> @ex-select2.sql
 SID SNAME        MAJOR YEAR BDATE
---- ------------ ----- ---- ------------
   8 Brenda       COMP     2 13-AUG-80
 1,0 Dupont       ENGL     1 13-MAY-80
   8 Brenda       COMP     2 13-AUG-80
 1,4 Jack         CSAP     1 12-FEB-77
```

The DUAL table in Oracle

```
SQL> describe dual;
 Name            Null?     Type
 ---------------- --------- ---------------
 DUMMY                      VARCHAR2(1)
```

Contains one row and one column. Can be used to put results

```
SQL> select power(2,10) from dual;

POWER(2,10)
-----------
       1024
```

select sysdate from dual;

```
SQL> select to_date(sysdate) from dual;

TO_DATE(S
---------
29-SEP-02
```

```
SQL> select add_months(sysdate,2) from dual;
ADD_MONTH
---------
29-NOV-02
```

Lets make Brenda younger

```
SQL> update student
set bdate=(select add_months(bdate,36)from dual)
where sid=8
```

```
update student
set bdate= add_months(bdate,36)
where sid=8
```

```
SQL> select * from student where sid=8;

 SID SNAME                MAJOR YEAR BDATE
---- -------------------- ----- ----
---------------
   8 Brenda               COMP     2 13-AUG-80
```

13-AUG-77

Editing SQL Buffer

Command	abbrev.	Operation on crnt. line/all li
append txt	a text	adds text at the end of a line
change /old/new/	c /old/new/	change old to new in a line
change /txt	c /txt	delete text from a line
clear buffer	cl buff	delete all lines in the buffer
delete	del	delete the current line
delete n	**del** n	delete line n
delete last	**del** last	delete the last line of the buffer
delete n,m	**del** n,m	delete lines n - m from buffer
ed	**ed**	edit the buffer or a file
get file		load file into buffer
input	i	add one or more lines
input txt	i txt	add text as a line
host		exit temp to OS, exit back to SQLPlus
list	l	list all lines of buffer
list n	l n (n)	list line n and make it current
list *	l *	list current. line

Editing SQL Buffer

ommand	abbrev.	Operation on crnt. line/all line
st last	l last	list last line
st m n	l m n	list lines m – n
ve file	sav file	save buffer to file
un	**/**	execute the commands in buffer

ther useful commands:

ter user *userid* **identified by** *newpasswor***d**

ool nameoffile

omments

for multi-line comments */

m for a single line comment

- comments that can start anywhere in a line up to the eol

```
create table student  -- we will create a table for students
(SID NUMBER(7) primary key not null, --not null is redundant
 SNAME VARCHAR2(20),  --varchar2 is a variable length string
 /*
  We will now define
  the student’s major and year
 */
MAJOR CHAR(4),
YEAR NUMBER(1),
rem BDATE is his/her birth date
rem It can be used to compute the age which is not stored.
BDATE DATE)
```

The editor used for the ed command is the default editor set using

setenv EDITOR {emas| vi | gedit | xemacs | ndedit} **for tcsh/csh**
export EDITOR={ emas| vi | gedit | xemacs | ndedit} **for bash**

Alternatively, you can set up your editor using the define command:

SQL> define _editor=emacs

```
SQL> define _USER=scott
SQL> define _PW=tiger
SQL> define
DEFINE _CONNECT_IDENTIFIER = "cind" (CHAR)
DEFINE _SQLPLUS_RELEASE = "902000100" (CHAR)
DEFINE _EDITOR        = "emacs" (CHAR)
DEFINE _O_VERSION      = "Oracle9i Enterprise Edition Release
9.2.0.1.0 - Production
With the Partitioning, OLAP and Oracle Data Mining options
JServer Release 9.2.0.1.0 - Production" (CHAR)
DEFINE _O_RELEASE      = "902000100" (CHAR)
DEFINE _RC             = "0" (CHAR)
DEFINE _USER           = "scott" (CHAR)
DEFINE _PW             = "tiger" (CHAR)
```

Show user defined varaibles

MySQL/Mariadb do not have, to date some of these interactive terminal based features

For most of the current versions of DB server have added web based functions

One can use phpMyadmin mySQLweb

Course Notes
Files and Databases
Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

A short introduction to ER & SQL

Bipin C. DESAI

Pl. see: https://users.encs.concordia.ca/~bcdesai/CopyForward.pdf

Database Languages

- A Database Management System (DBMS) provides two types of languages; they may also be viewed as components of the DBMS language:
  - Data Definition Language (DDL)
    - Language (notation) for *defining* and modifying a database schema
    - It includes syntax for declaring tables, indexes, views, constraints, etc.)
  - Data Manipulation Language (DML)
    - Language for *accessing* and *manipulating* the data (organized/stored according to the appropriate data model)

Query Languages

- Theoretical:
  Relational Algebra, Relational Calculus, Datalog
- Commercial: SQL
- First there were two: SEQUEL (Ingres) and SQL(R)
- SQL developed originally at IBM in 1976
  - First standard: SQL-86
  - Second standard: SQL-92
  - Latest standard: SQL-99, or SQL3,
    SQL3 standard has over 1,000 pages of document
- SQL is the de-facto standard for RDBMS
- The SQL query language components:
  - DDL (e.g., create)
  - DML(e.g., select, insert, update, delete)

Simple SQL Queries

A SQL query has a form**:**

SELECT . . .

FROM . . .

WHERE . . . ;

The **SELECT** clause defines the schema of the result

The **FROM** clause gives the source relation(s) of the query

The **WHERE** clause is one or more predicates to ‘select” the tuples of interest.

The query result is a relation and it is **unnamed**.

Example “theoretical” SQL Query

Relation schema:

Course (Cno, Cname, credits)

Query in natural language (English):

Find all the courses stored in the database

Query in SQL:

SELECT *

FROM Course;

Here the “ * “ in “**SELECT** *” means **all** attributes in the **relation(s)** involved.

More Examples SQL Query

- Relation schema:

  Movie (title, year, length, filmType)

Query in natural language (English):

Find the titles of all movies stored in the database

Query in SQL:

SELECT title
FROM Movie;

Relation schema:

Student (SID, FirstName, LastName, Address, GPA)

Query in natural language (English):

Find the SID of every student whose GPA is greater than 3

Query in SQL:

SELECT SID
FROM Student
WHERE GPA > 3;

The “WHERE” clause

The expressions that may follow **WHERE** are conditions

Standard comparison operators **θ** includes { =, <>, <, >, <=, >= }

The values that may be compared include constants and attributes of the relation(s) mentioned in **FROM** clause

Simple expression

A θ Value

A θ B

Where **A**, **B** are attributes and

θ is a comparison operator

We may also apply the usual arithmetic operators, +,-,*,/, etc. to numeric values before comparing them

(year - 1930) * (year - 1930) < 100

The result of a comparison is a Boolean value **TRUE** or **FALSE**

Boolean expressions can be combined by the logical operators **AND**, **OR**, and **NOT**

- Relation schema: **Movie** (title, year, length, filmType)
- Query: Find the titles of all color movies produced in 1950
- Query in SQL:

```
SELECT title
FROM Movie
WHERE filmType = ’color’ AND year = 1950;
```

- Query: Find the titles of color movies that are either made after 1970 or are less than 90 minutes long
- Query in SQL:

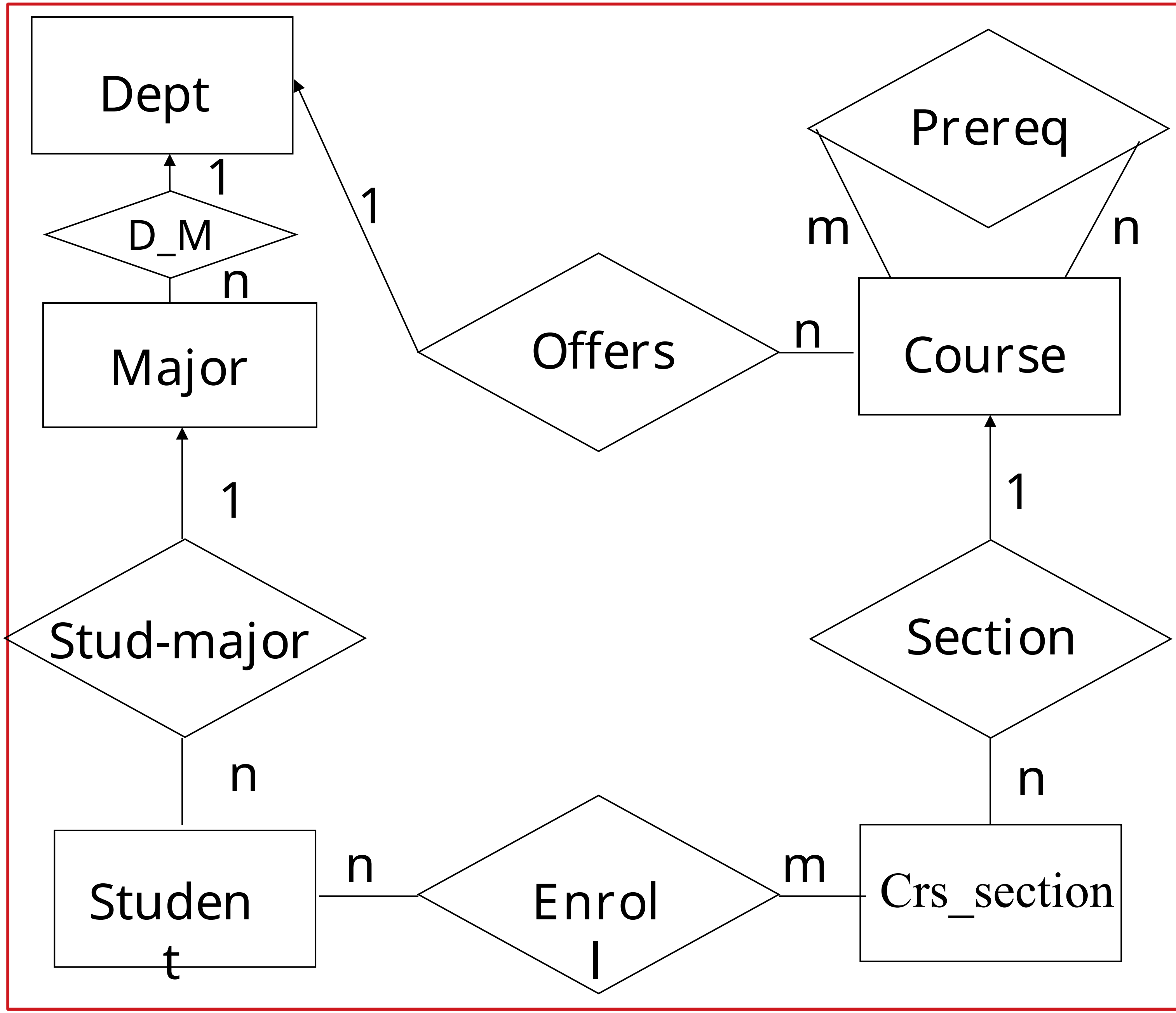
```
SELECT title
FROM Movie
WHERE (year > 1970 OR length < 90) AND filmType =
   ’color’;
```

Note the precedence rules, when parentheses are absent:

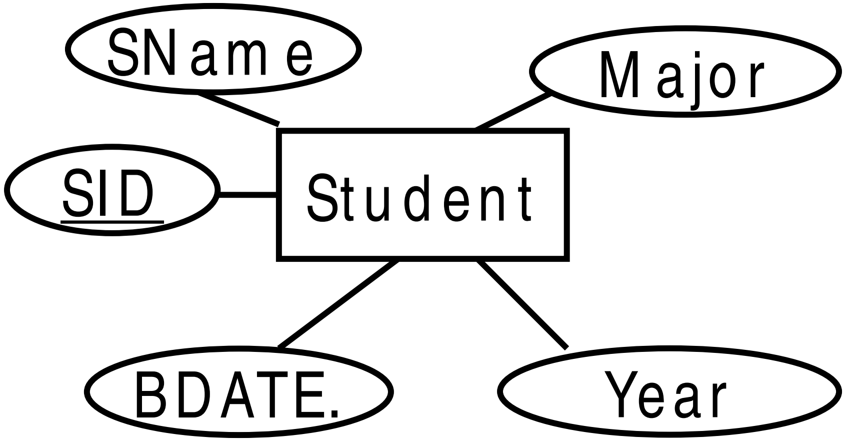
AND takes precedence over **OR**,

and **NOT** takes precedence over both

An example of using SQL from E-R to RDBMS


Dept
1
D_M
n
Major
1
Offers
1
n
Prereq
m
n
Course
1
Section
n
Stud-major
n
Studen
t
n
Enrol
l
m
Crs_section

SQL DDL example:



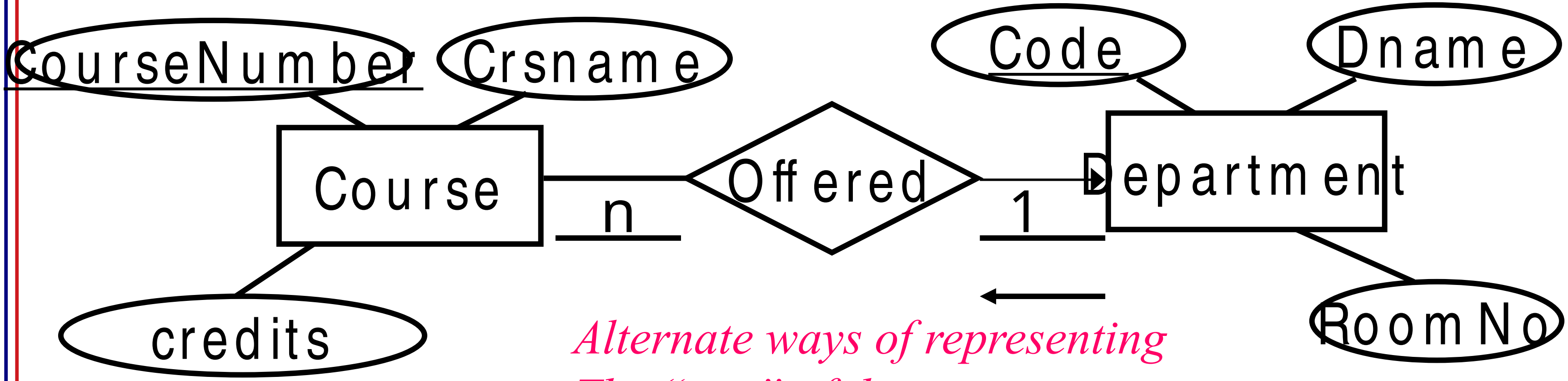
```
create table student
(SID NUMBER(3) primary key not null,
 SNAME VARCHAR2(20),
 MAJOR CHAR(4),
 YEAR NUMBER(1),
 BDATE DATE)
/
```

```
insert into student values
(8,'Brenda','COMP','2','13-AUG-77');
(8,'Brenda','COMP',',‘1977-08-13');
insert into student values
(10,‘Mary','ENGL','1','13-MAY-80');
(10,‘Mary','ENGL','1',‘1980-5-13’);
insert into student values
(13,'Keily','SENG','4','12-AUG-80');
insert into student values
(14,‘Jack','CSAP','1','12-FEB-77');
```

Many to one relationship

A department offers many courses
A given course can be offered by only one department

There is no standard regarding the direction of arrow for the "one" entity.

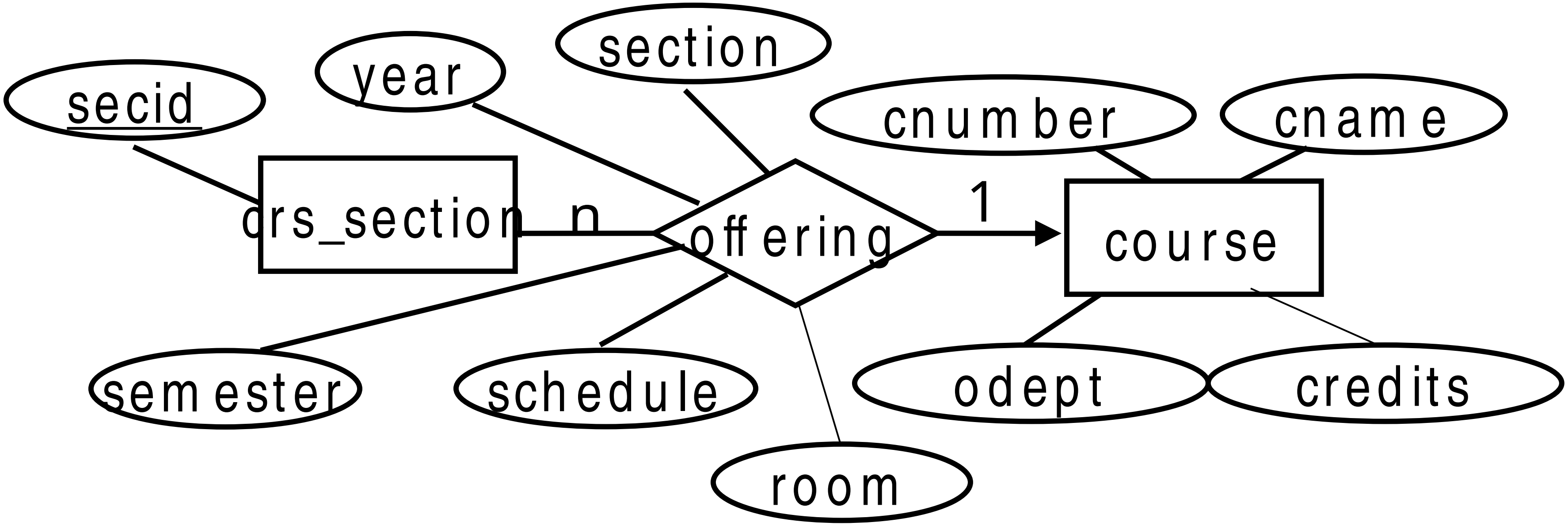


Alternate ways of representing The "one" of the many-to-one relationship; arrow either pointing to entity on the "one side" or pointing to the relationship

```
create table course
(COURSE_NAME CHAR(20),
 COURSE_NUMBER CHAR(8) primary key NOT
NULL,
 CREDIT_HOURS NUMBER(2),
 OFFERING_DEPT CHAR(4))
 tablespace TUTOR pctfree 5
/
```

Note how the relationship w/o an attribute is “merged” with one of the entity

```
insert into course values
('C++','COMP248',3,'COMP');

insert into course values
('DATA STRUCTURES ','COMP352',3,'COMP');

insert into course values
('OPERATING SYSTEMS','COMP346',4,'COMP');

insert into course values
('DATABASE','COMP353',4,'COMP');
```



NB: Here course section (crssection) is really a "weak" entity; However, in most cases it is promoted a "strong" entity by introducing an identifying key attribute section ID (secid)

```
create table crs_section
(SECID NUMBER(6) primary key NOT NULL,
 COURSE_NUM CHAR(8),
 SECTION CHAR(2),
 SEMESTER CHAR(4),
 YEAR CHAR(4),
 SCHEDULE CHAR(10),
 ROOM CHAR(7))
 tablespace TUTOR pctfree 2
/
```

Note: We have replaced an entity and the relationship With a single relation

```
insert into crs_section values
(85,'COMP352','A','FALL', '1998','TH16001715','H123');
insert into crs_section values
(90,'COMP353','B','FALL','1999','MW08451000','H631');
```

```
create table enrolment
(STUDENT_NUMBER NUMBER(3) not null,
 SECTION_ID NUMBER(6) not null,
 GRADE CHAR(1),
 primary key(student_number, section_id))
tablespace TUTOR pctfree
/
            insert into enrolment values(8,85,null);
            insert into enrolment values(10,90,null);
            insert into enrolment values(8,90,null);
            insert into enrolment values(14,90,null);
```

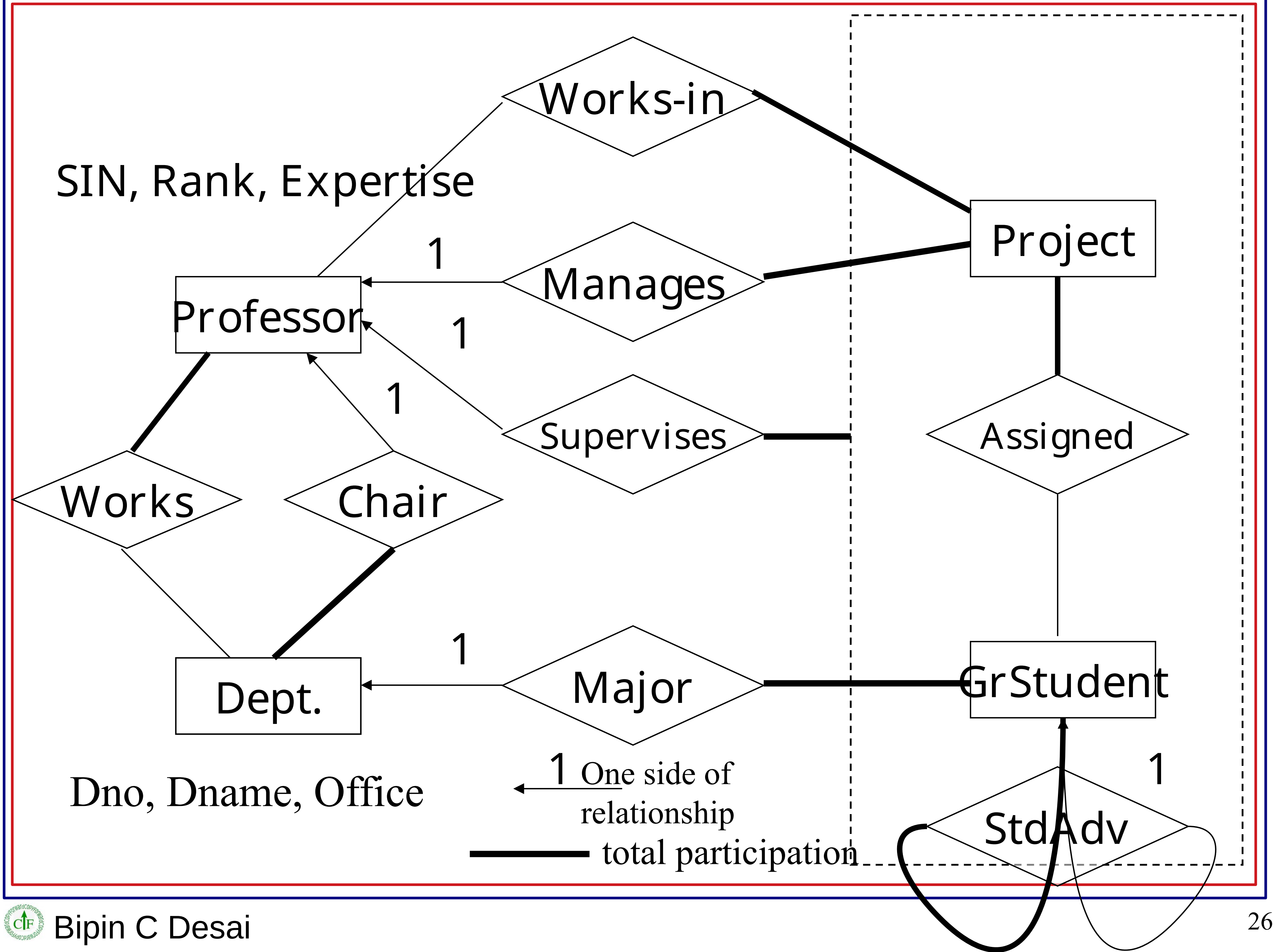
Find details of studs. taking a course offered by the “DISC” dept.

```
select s.SID, s.SNAME, s.MAJOR, s.YEAR, s.BDATE
from student s, dept d, course c, crs_section r, enrolment e
where  c.ODEPT=d.CODE and
       r.COURSE_NUM=c.CNUMBER and
       r.SECID=e.SECTION_ID and
       e.STUDENT_NUMBER = s.SID and
       d.CODE= 'DISC';
```

```
SID SNAME        MAJOR YEAR BDATE
---- ------------ ----- ---- ------------
 1,4 Jack         CSAP      1 12-FEB-77
```

More examples of using E-R modeling

Professors have a SIN, a name, an age, a rank, and a research specialty.
Projects have a project number, a sponsor name (e.g., NSERC),
a starting date, an ending date, and a budget.
Graduate students have a SIN, a name, an age, and a degree program
(e.g., M.S. or Ph. D.).
Each project is managed by a professor (principal investigator).
Each project is worked on by one or more professors (co-investigators).
Professors can manage and/or work on multiple projects.
Each project is worked on by one or more graduate students (research
assistants).
When a graduate student works on a project, is supervised by a
participating professor.
Graduate students can work on multiple projects.
Departments have a department number, a department name, and a main office.
Departments have a chairman who runs the department.
Professors work in one or more departments(%time)
Graduate students have one major department for their degree.
Each graduate student has another, more senior graduate student
(student advisor)



Works-in

SIN, Rank, Expertise

Project

1

Manages

Professor

1

1

Supervises

Assigned

Works

Chair

1

Major

GrStudent

Dept.

Dno, Dname, Office

1 One side of relationship

1

StdAdv

total participation

In a company database, you need to store information about

employees (SIN,salary and phone),

departments (dno, dname, budget), and

children of employees (with name and age as attributes).

Employees work in departments;

Each employee works in only one department;

A department could have many employees;

Each department is managed by an employee;

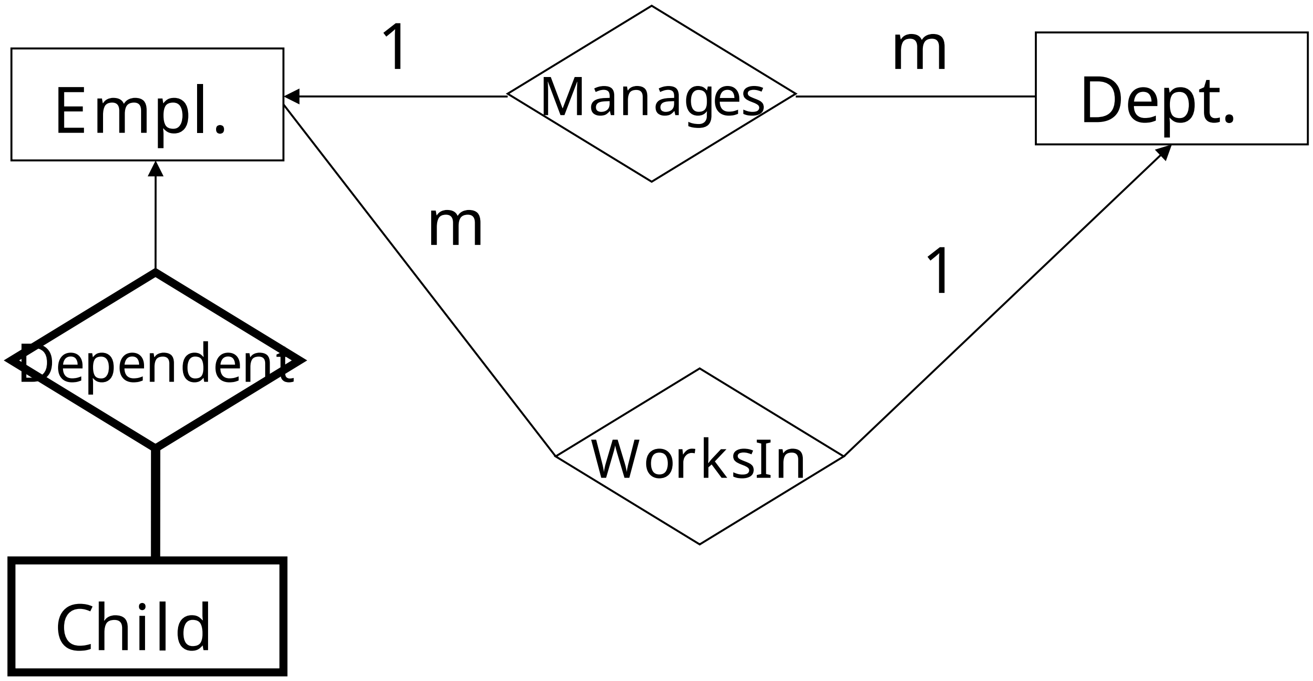
Each department has only one manager(an employee);

A manager could manage many departments

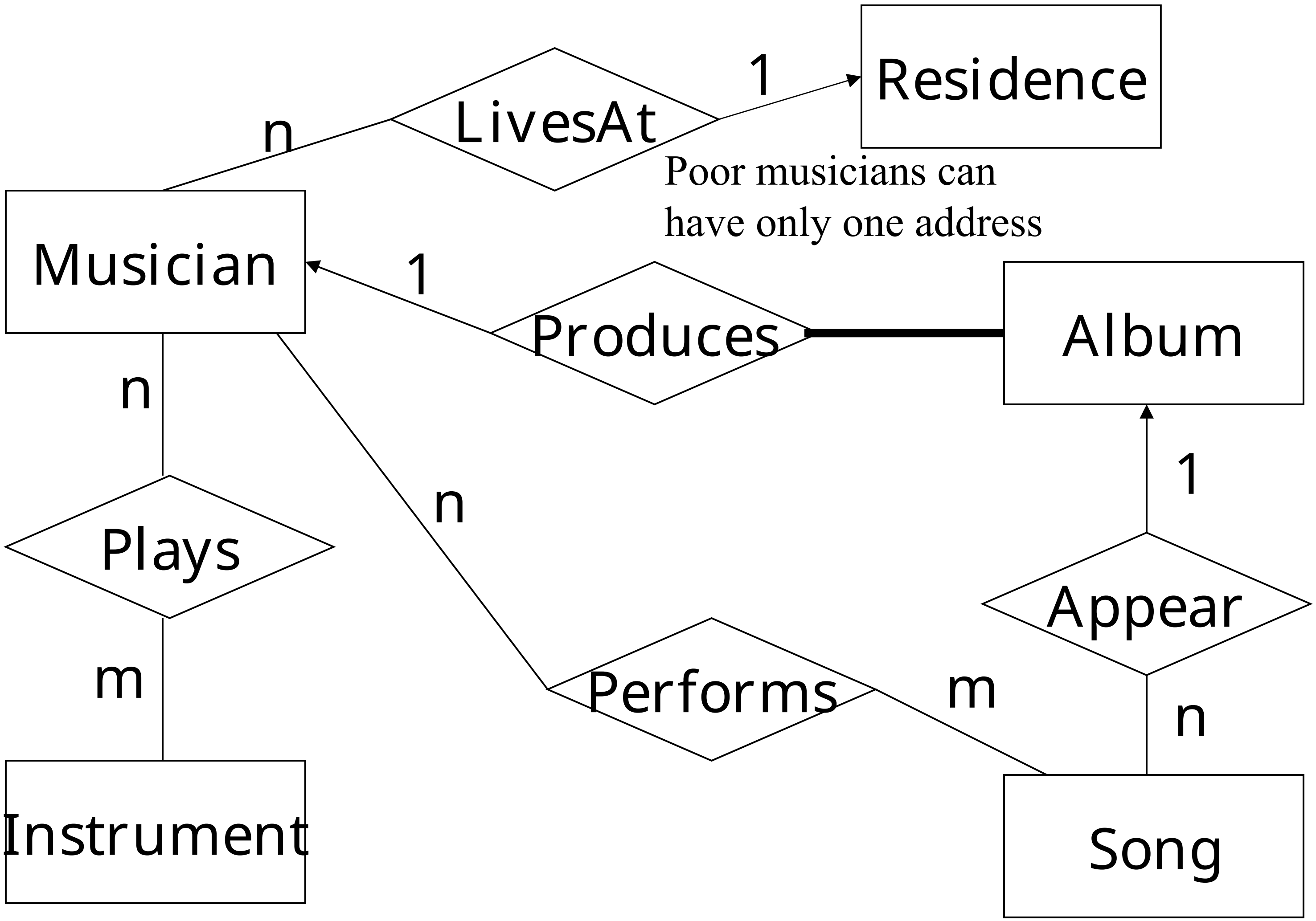
A child can only be identified by name

An employee has only one child with a given name

only one parent can declare a child as a dependent

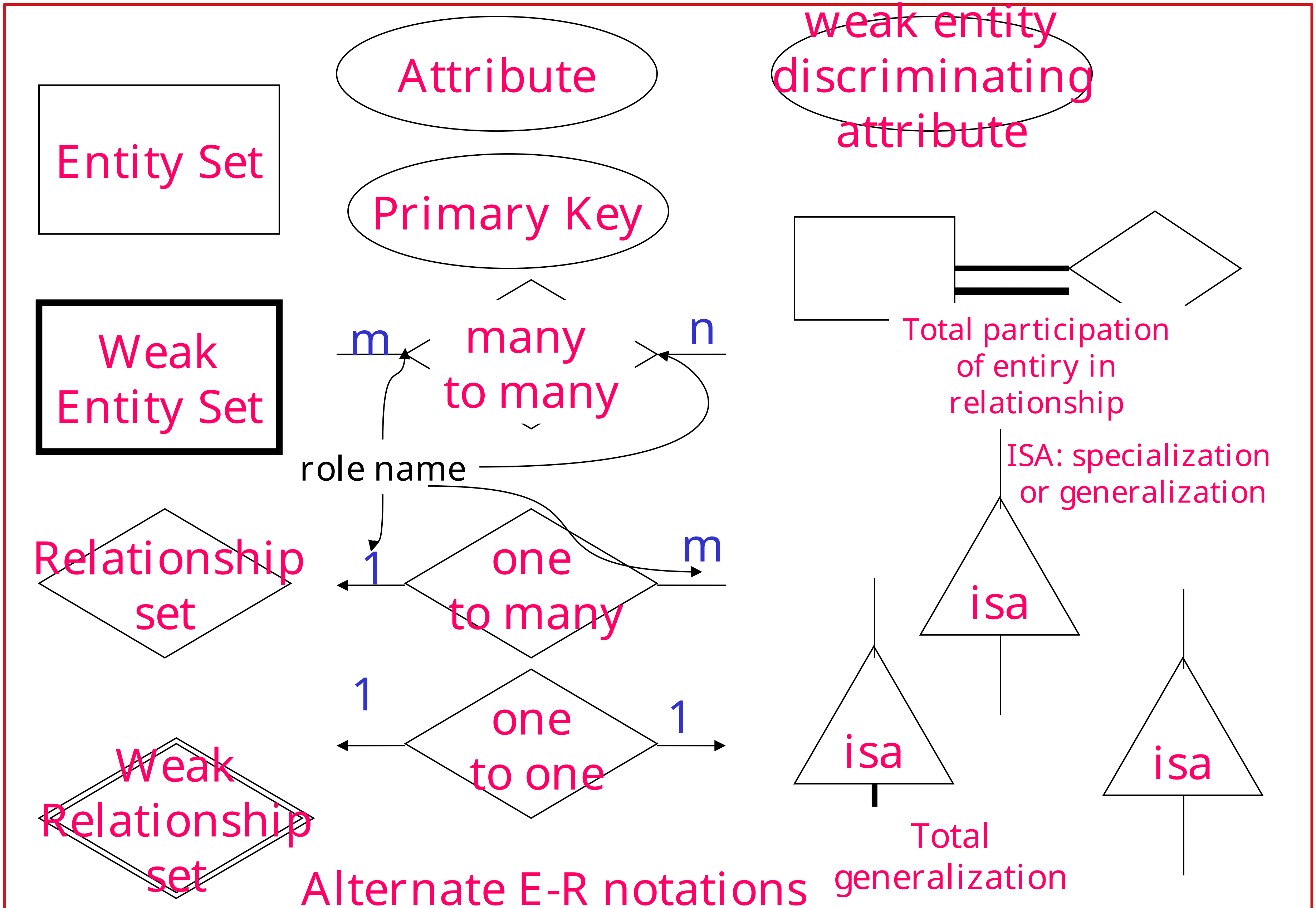

1
Manages
m
Empl.
Dept.
m
1
Dependent
WorksIn
Child

Each musician that records at Notown has an SIN, a name,
an address, and a phone number.
Musicians often share the same address,
no address has more than one phone.
Each instrument that is used in songs recorded at Notown has a name and
a musical key
Each album that is recorded on has a title, a copyright date, and
an album identifier.
Each song recorded has a title and an author.
Each musician may play several instruments, and
a given instrument may b e played by several musicians.
Each album has a number of songs on it,
but no song may appear on more than one album.
Each song is performed by one or more musicians, and
a musician may perform a number of songs.
For each album, there is exactly one musician that acts as its producer.
A musician may produce several albums.



Poor musicians can have only one address

Notes: Since a songs must appear on only one album, Appear is a many to one relationship. Similarly for Produces. Album requires total participation in Produces. Some songs may not be recorded and there may be some instruments that nobody can play!



Entity Set

Attribute

weak entity discriminating attribute

Primary Key

Weak Entity Set

m

many to many

n

Total participation of entiry in relationship

role name

ISA: specialization or generalization

Relationship set

1

one to many

m

isa

1

one to one

1

isa

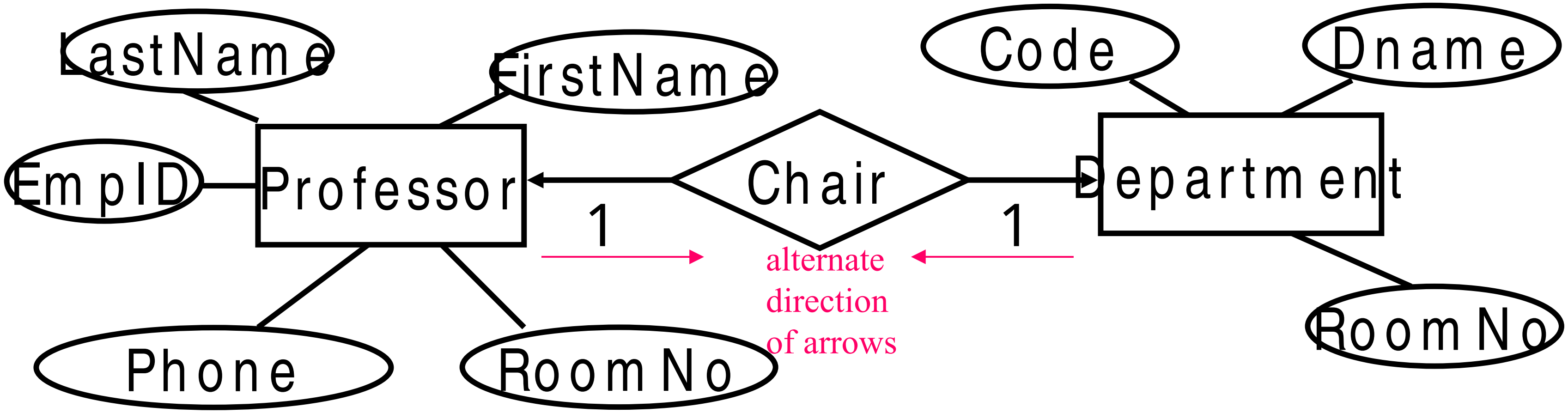
isa

Weak Relationship set

Total generalization

Alternate E-R notations

- A **one-one** relationship between **Department** and **its** chair (a dept. has one chair and a prof. is the chair of at most one dept) is represented by
  - arrows pointing to both Department and Professor or
  - indicated by a line with the number 1 on it.
  - Sometimes the arrow is in the opposite direction(pointing to the diamond)



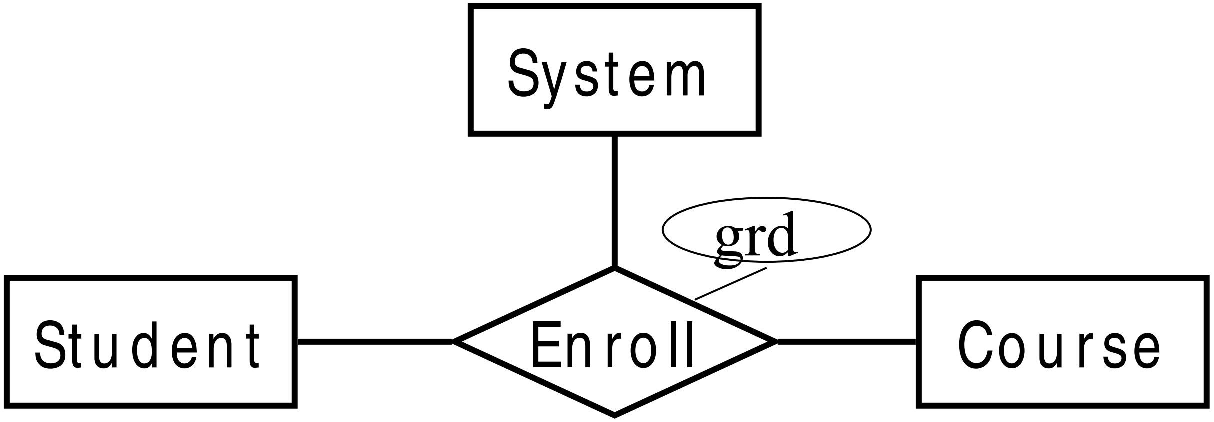
alternate direction of arrows

ODL allows only **binary** relationships, i.e., relationships involving two classes.

E/R model makes it convenient to define **(n-ary)** relationships – relationships involving n entity sets

A **n-ary** relationship in an **E/R** diagram is represented by lines from the relationship (diamond) to each of the participating entity sets (rectangles).

N-ary Relationships

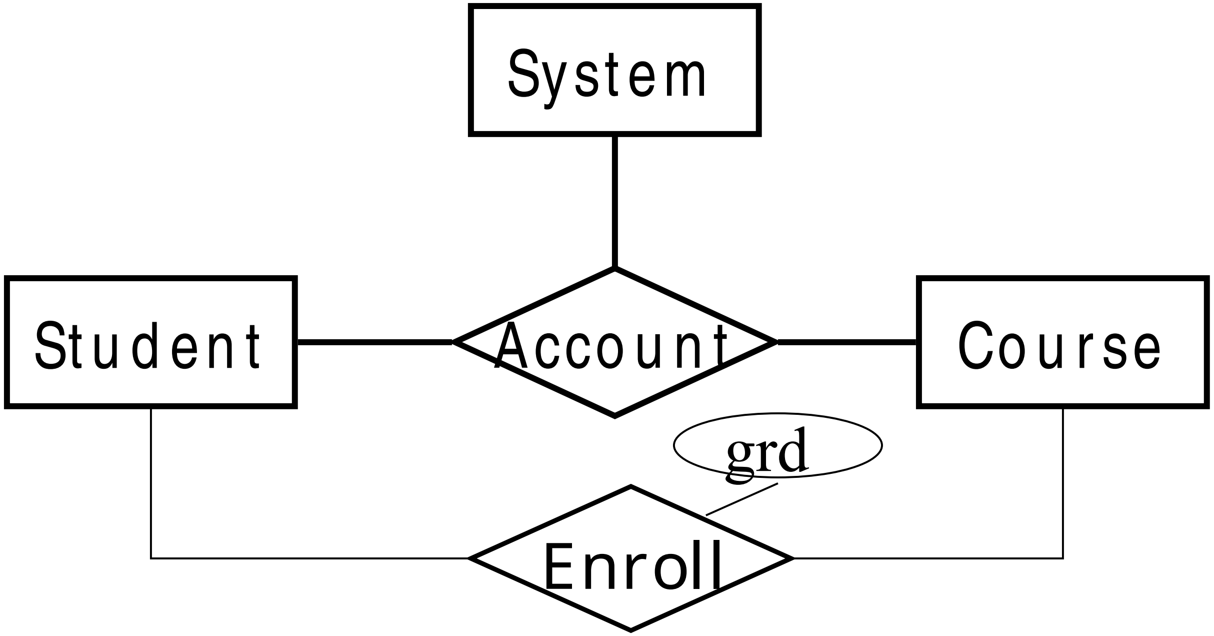


Multiplicity of this ternary relationship: n to n to n

Enroll(sid(fk →Student), cno(fk → Course),, sys(fk → System), grd)

What is the problem here?
What is the schema for Enroll?
How is the n:n:n relationship mapped to a relation(table)?

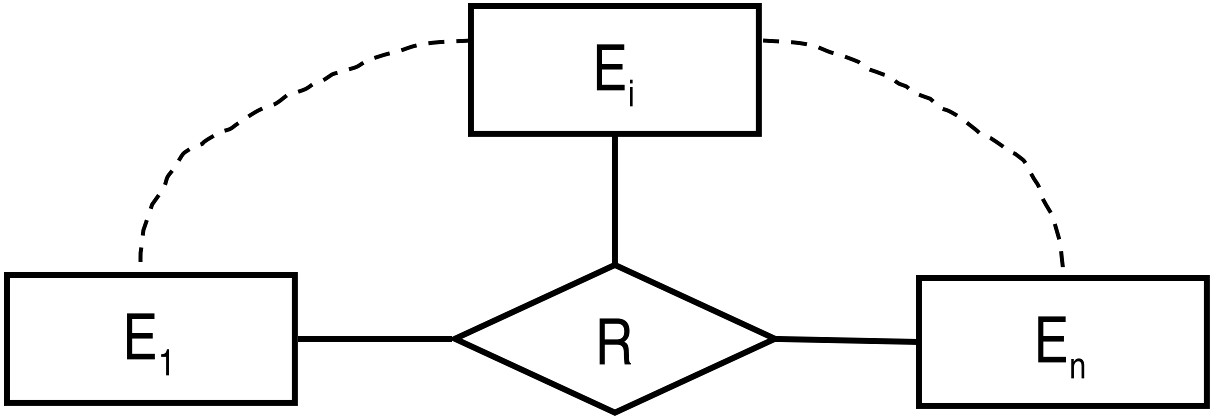
N-ary Relationships



Multiplicity of this ternary relationship: n to n to n

Account(**sid**(fk →Student), **cno**(fk → Course), **sys**(fk → System), **uid**)
Enroll(**sid**(fk →Student), **cno**(fk → Course), grd)

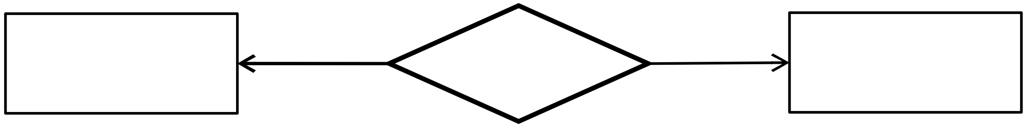
N-ary Relationships



Multiplicity of this N-ary relationship: n_1---- n_i ----- n_n

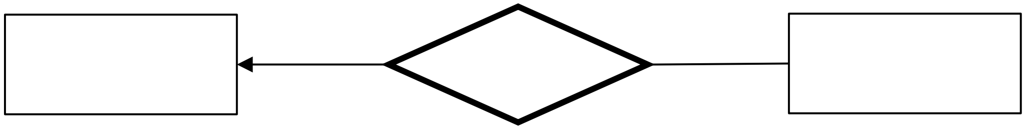
Any of this n_i could be 1

Any of this could be multiple



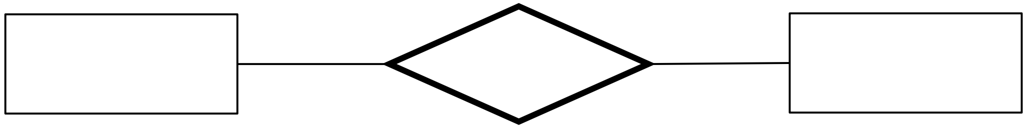
If a binary relationship between two entity sets is 1-to-1,

- the primary key of the relationship is the key of the entity from either of the '1' side, the other side is a foreign key*(would be unique)*



If a binary relationship between two entity sets is 1-to-many,

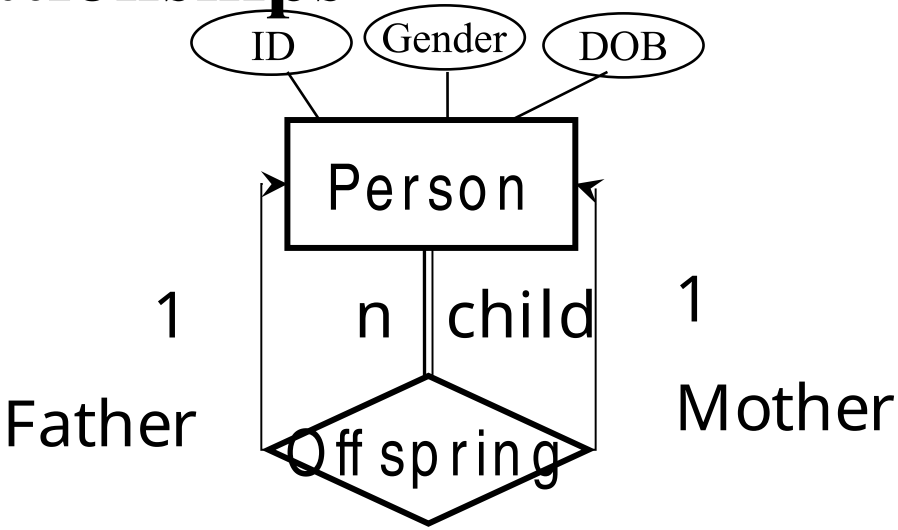
- the primary key of the relationship is from the 'm' side, the '1' side is theforeign key *(would be unique)! This 'convention' may be reversed for convenience -specially if the number of entities on the one side is much smaller!*



If a binary relationship between two entity sets is m-to-n,

- the primary key is composite, consisting of the primary key of the entities from each side of the relationship

N-ary Relationships



Multiplicity of this ternary relationship: 1 to 1 to n

Offspring(fid (fatherID → Person(ID)), motherID (fk → Person(ID)), ChildID (fk → Person(ID)))

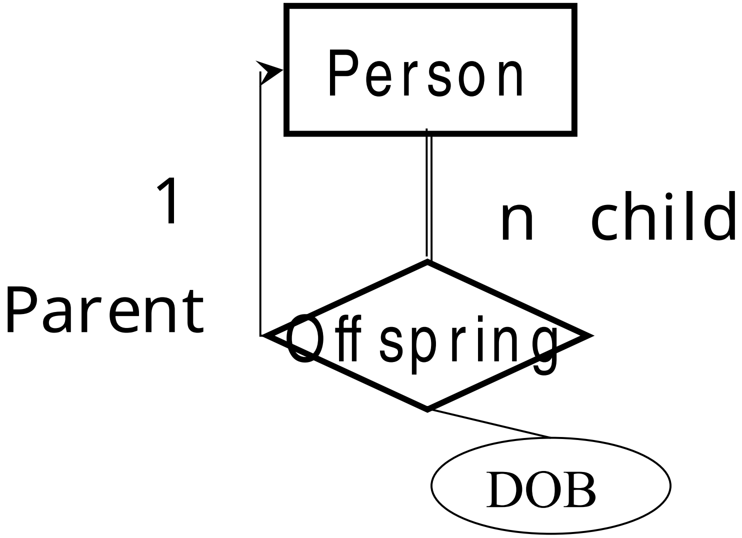
Who here is the father and mother?
What is the key?

```
create table person(
ID number primary key,
gender char(1),
DOB date);
insert into person values(1, 'M', '11-Jan-1900');
insert into person values(2,'F', '11-Jan-1902');
insert into person values(121,'M', '11-Jan-1925');
insert into person values(122,'F', '11-Jan-1927');
insert into person values(3,'M', '11-Jan-1901');
insert into person values(4,'F', '11-Jan-1903');
insert into person values(341,'M', '11-Jan-1926');
insert into person values(342,'F', '11-Jan-1928');
insert into person
          values(1213421,'M', '11-Jan-1948');
insert into person
          values(1213422,'F', '11-Jan-1950');
```

Could two tuples exist in offspring with the same cid???

```
create table offspring(
fid number, mid number,
cid number primary key,
foreign key (fid)
          references person(id),
foreign key (mid)
          references person(id),
foreign key (cid)
          references person(id));
insert into offspring values(1,2,121);
insert into offspring
          values(1,2,122);
insert into offspring
values(3,4,341);
insert into offspring
          values(3,4,342);
insert into offspring
values(121,342, 1213421);
insert into offspring
values(121, 342, 1213422);
```

Replacing a ternary relation by a binary relation



What is the schema for Offspring here?

What is a possible inconsistency problem?

Who is the father, mother??

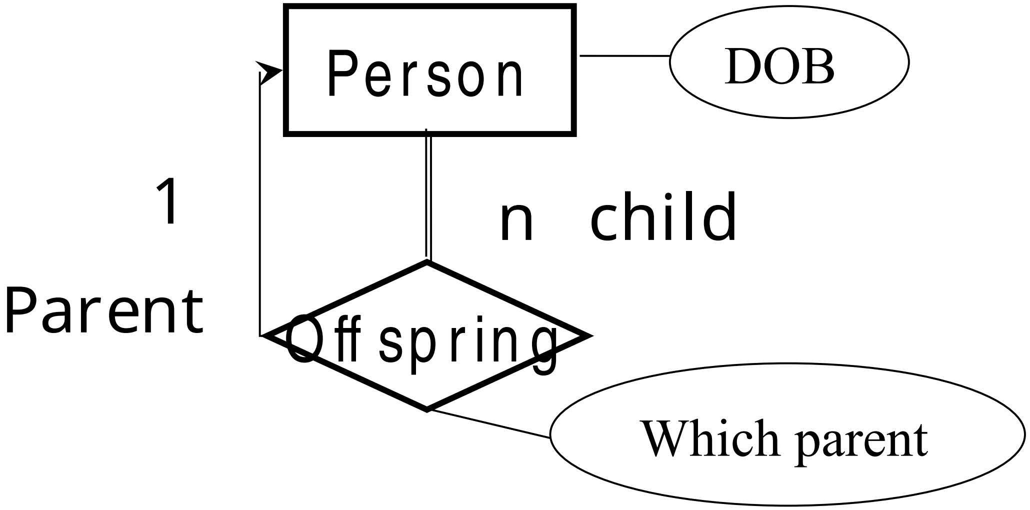
Offspring (IDC, IDF, IDM, DOB)

Father(IDC, IDF, DOB)

Mother(IDC, IDM, DOB)

- ***not the same ER***
- ***duplication of DOB***
- ***Composite key***

Replacing a ternary relation by a binary relation --- an alternate *non-normal form*



What is the schema for Offspring?
Is there a duplication problem?
What is the primary key?

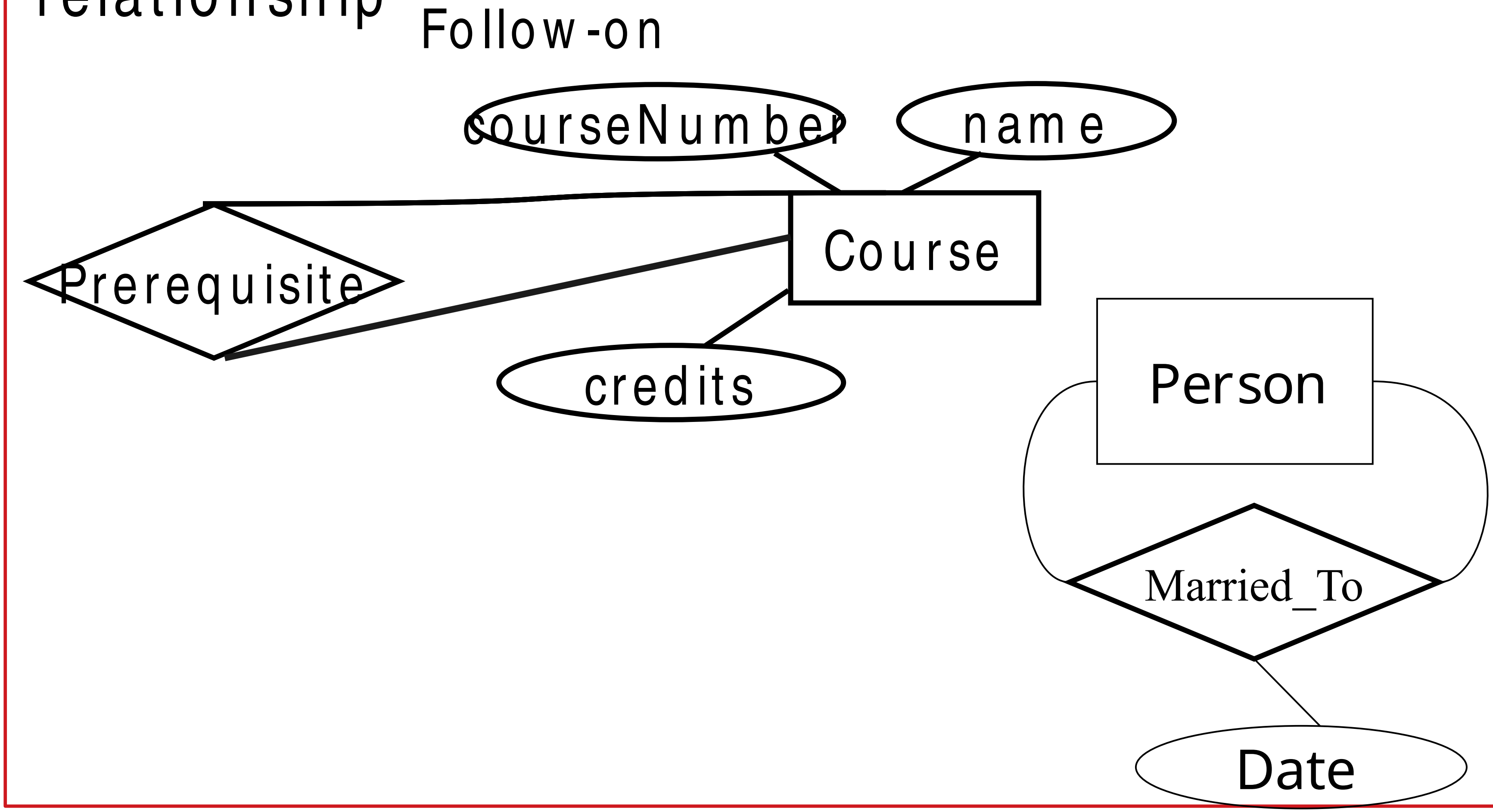
Multivalued attribute

121	1	Father
	2	Mother
122	1	Father
	2	Mother

Roles in Relationships

- It is possible that the same entity set appears **two** or **more** times in a relationship
- Suppose, we want to capture the relationship between two **courses**, one of which is the **pre-requisite/follow-on** of the other

Each line to the entity set represents a different **role** that the entity set plays in the relationship

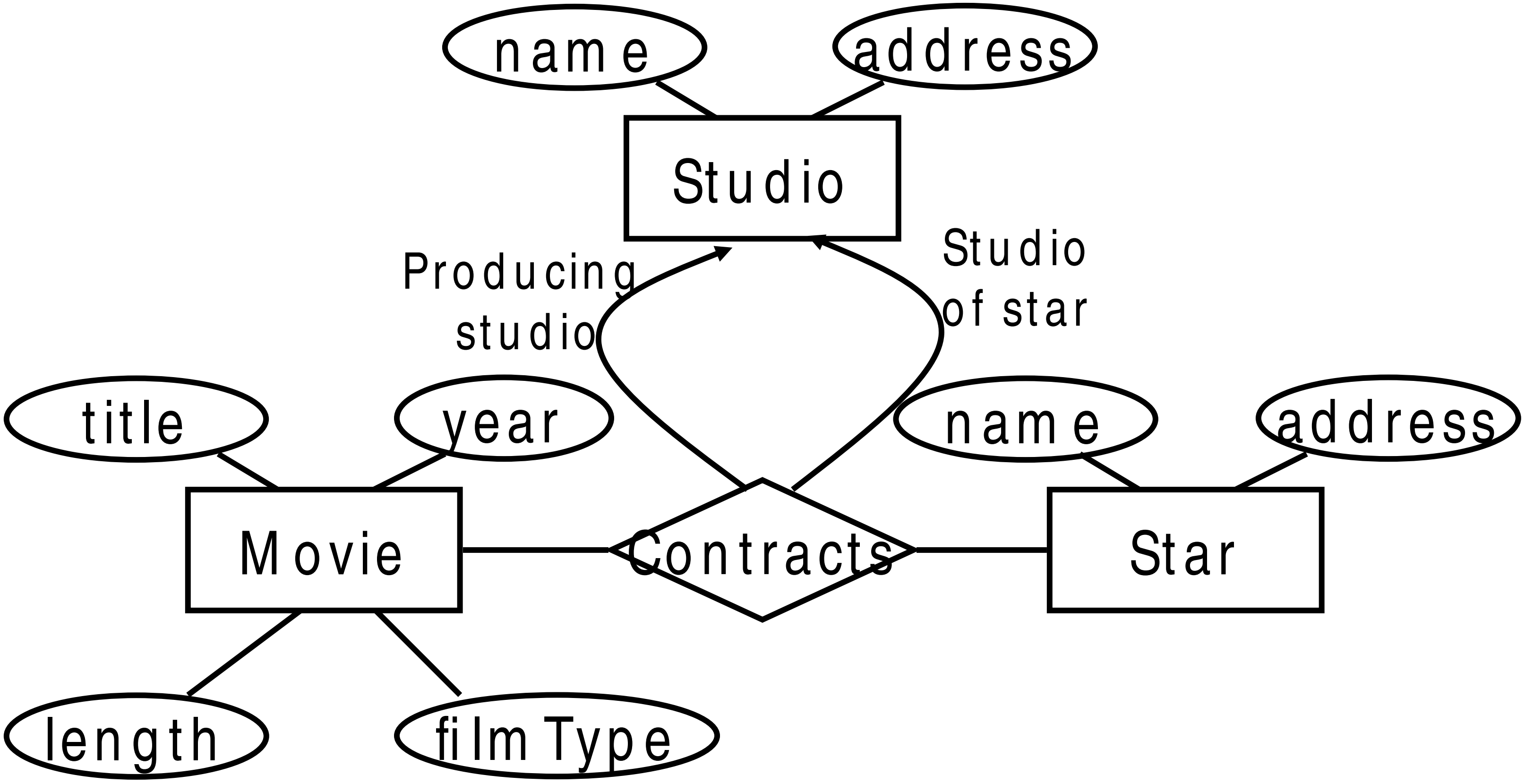


```
create table prereq
(COURSE_Number CHAR(8),
 PREREQ CHAR(8),
 primary key (course_number, prereq))
tablespace TUTOR pctfree 2
/
```

```
insert into prereq values('COMP353','COMP352');
insert into prereq values('COMP353','COMP346');
insert into prereq values('COMP352','COMP248');
```

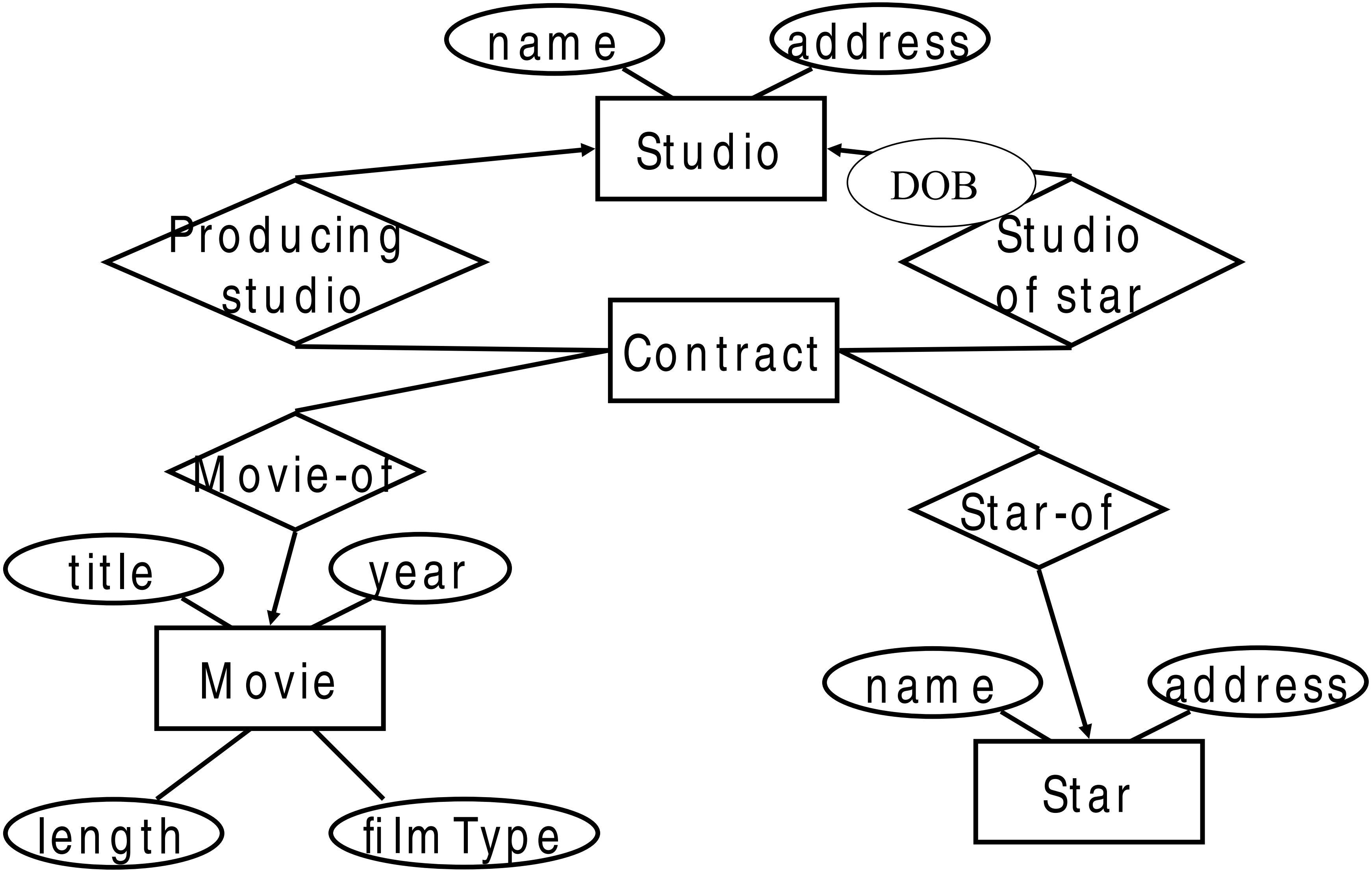
Suppose, each star is under contract with a single studio

The studio of the star may enter into a contract with another studio to allow that star to act in a particular movie

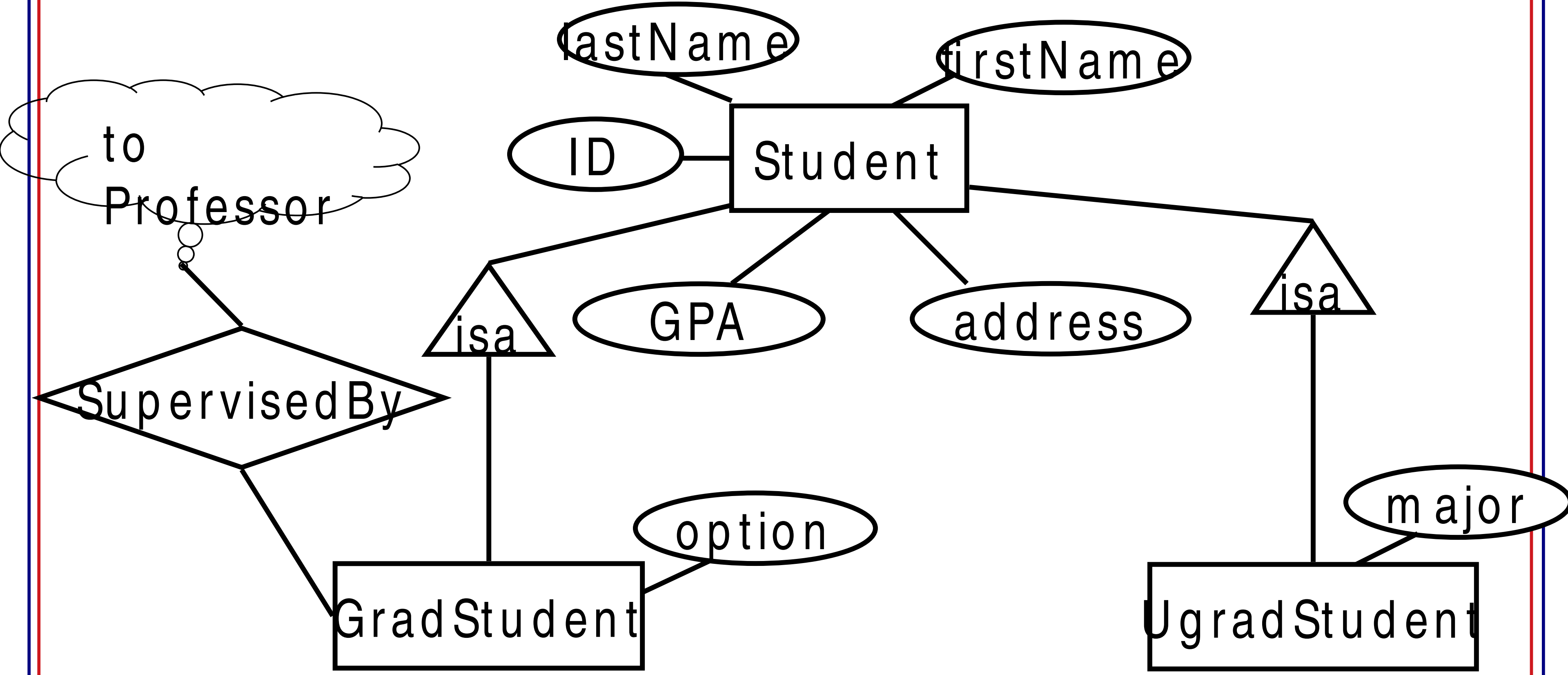


Converting n-ary relationship

- Any n-ary relationship may be converted into a **collection** of **binary** relationships **without loosing** any information???
  - Introduce a **new** entity set – **connecting** *existing* entity set – whose entities might be thought of as tuples of the relationship for the n-ary relationship
  - Introduce **many-to-one** relationships from the **connecting** entity set to each of the entity sets participating in the original n-ary relationship
  - If an entity set plays **more than one** role, then it is the target of one relationship for **each role**
- Usually doesn't convey the same semantics – limitation of modeling


name
address
Studio
DOB
Producing studio
Studio of star
Contract
Movie-of
Star-of
title
year
Movie
name
address
length
film Type
Star

Inheritance in E/R is expressed by ***isa*** relationship



- There is a subtle difference between the concept of inheritance in ODL and in the E/R model
- In **ODL**, an object must be a member of exactly one class
- In the **E/R** model
  - We shall view an entity as having “components” belonging to several entity sets that are “part of” a single **isa**-hierarchy
  - The “components” are connected into a single entity by the **isa** relationships
  - The entity has whatever attributes any of its components has, and it participates in whatever relationships its components participate in

Ⓟ We need to represent an entity (e.g., CartoonMurderMystery) in the diagram only if it has attributes and/or relationships of its own

Constraints

- There are some important **aspects** of the **real world** that **cannot** be represented using the **ODL** or **E/R** model introduced so far
- The additional information about these aspects often takes the form of **constraints** on the data
- Sometimes modeling this additional information goes beyond the **structural** and **type** constraints imposed by **classes**, **entity sets**, **attributes,** and **relationships**

Keys are (sets of) attributes that **uniquely** identify an object within its class or an entity within its entity set;

K ⊆ R. ***no two entities may agree in all their key values***

Single-value constraints are requirements that the value of an attribute be unique. In addition to key constraints, other attributes must a have a single-value constraints.

Also in an "one" relationship

Referential integrity constraints are requirements that a value referred to by some object must actually exist in the database;

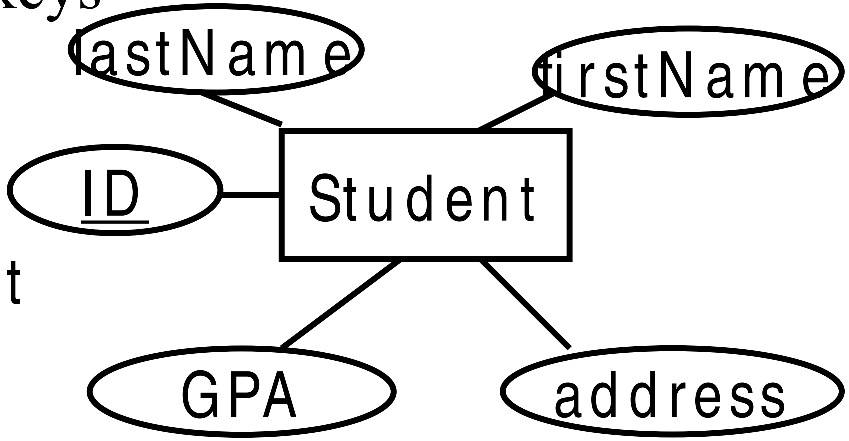
This means, no dangling pointers.

Domain constrains require that the value of an attribute must be drawn from a specific set of values (called attribute domain), or lies within a specific range

General constraints – arbitrary assertions that must hold on the DB

Keys

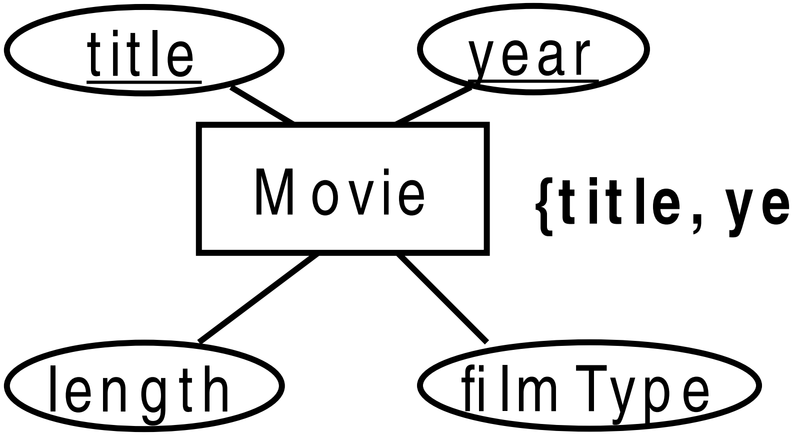
- A *super key* is a set of attributes whose values uniquely identify an entity in the entity set; this set may not be minimal.
- A **minimal super key** is called a **(***candidate***) key.**
- An entity may have more than one **key**. One of them is picked as the primary key**;** others may be called *alternate keys*
- In **E/R,** we underline the key attribute(s) of an entity set (i.e., those attributes that form the primary key of the set)
- No notation in E/R for alternate keys



ID is the key for the entity set **Student**

Example

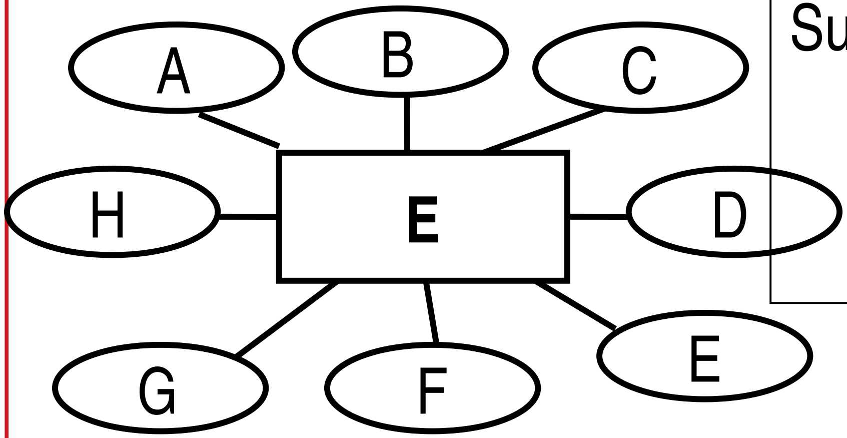
- What should we select as a key for **Movie** ?
- **title**?
  - there could be different movies with the same name
- **{title, year}**?
  - there still could be two movies made in the same year, with the same title, but that's very unlikely



{title, year} is a key for **Movie**

https://en.wikipedia.org/wiki/Lists_of_film_remakes

Selecting A Primary Key



Suppose candidate keys for **E** are :

1. **{A, B}**
2. **{D, E}**
3. **{F, G, H}**

- Which of the three should "we" pick as the **primary key**?

Criteria to choose a **primary key** when there are more than one candidate:

Total size

Number of attributes

Convenience

A combination of the above

Single Value Constraint

- In **E/R:**
  - attributes are **atomic**
  - an arrow (→) can be used to express the multiplicity
  - What about multi-valued attributes or relationships?

With the E/R model introduced so far, we cannot express the following options regarding the value of a single-valued **attribute**:

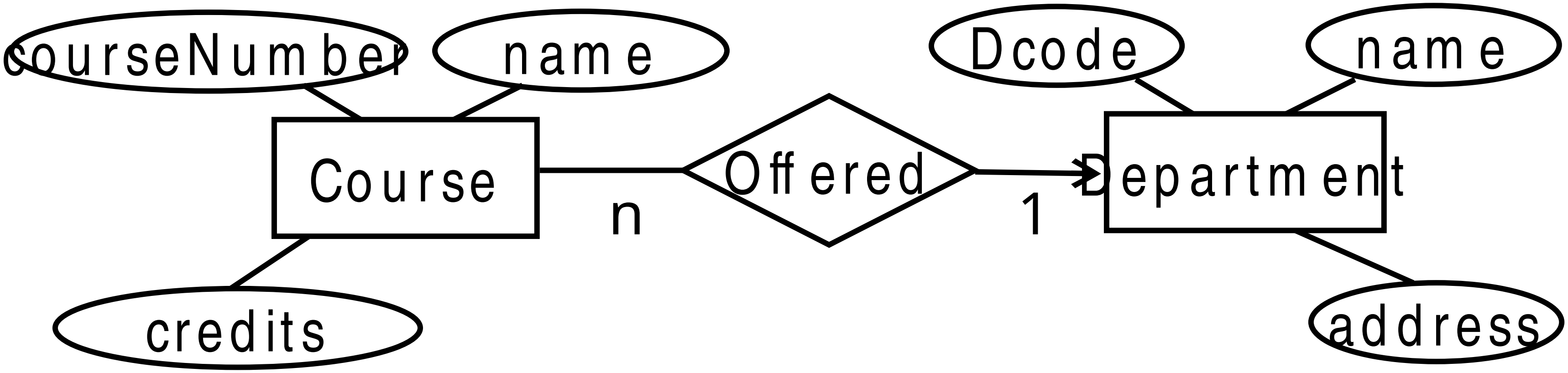
- Require that the value of that attribute to be present(not null)
- Or the presence of the value be optional (null allowed)

If the choice is not explicit, then we may conclude:

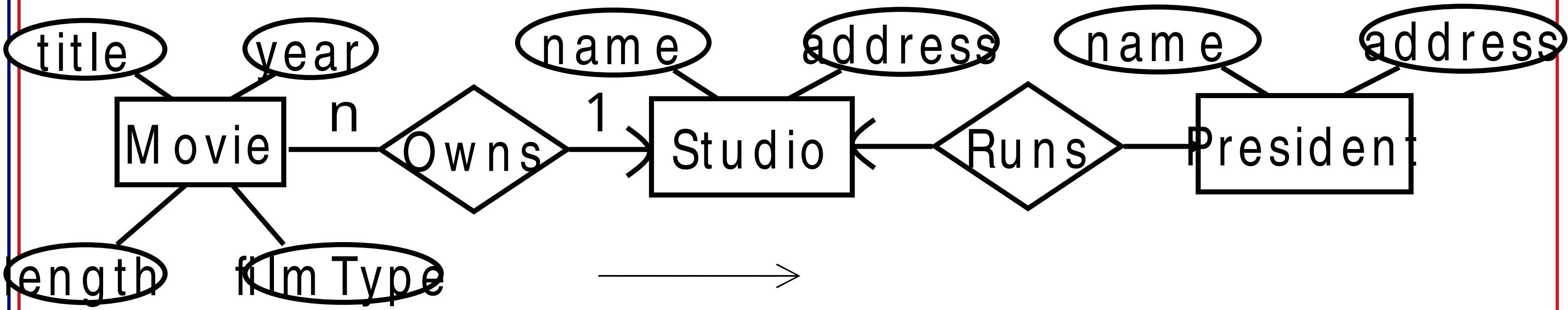
- The value must exist if an attribute is part of the key
- The value is optional, otherwise

Referential Integrity Constraints

- For relationships:
  - **Single-value + Existence = Referential Integrity Constraint**
- We extend the **arrow** notation to indicate a reference is mandatory (to support referential integrity)
  - The department that gives a course must always exist in the **Department** entity set



Referential Integrity Constraints



open arrow to denote ref. intg.

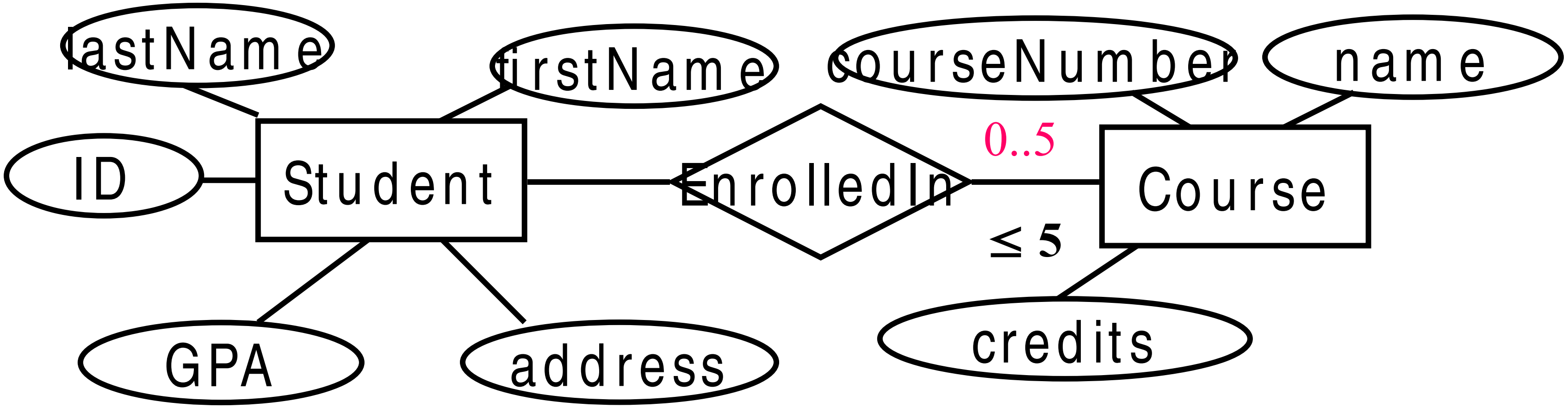
- The studio owning a movie must always be present in the Studios (extent of the Studio entity set)
- If a president runs a studio, then that studio exists in the Studios
- There could be studios without a president (temporarily)

Domain constraints

- **Domain constraints** restrict the values of an attribute to be drawn from a set
  - In ODL, we give a type to the attributes and hence limit their set of values
  - ODL does **not** support other restrictions, such as that the value should be within a certain range
  - E/R, in general, does **not** support imposing domain constraints

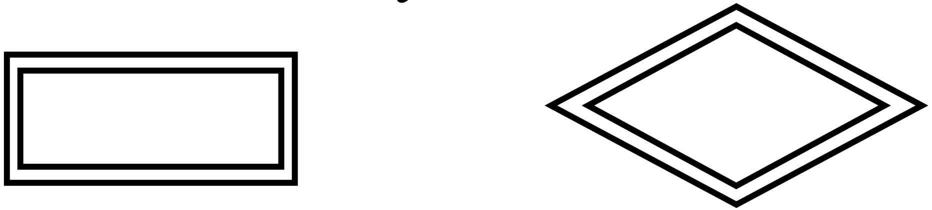
Relationship degree constraints

- **Relationship degree constraints** restrict the number that an entity/object can participate in a relationship
  - For example, we can impose a constraint saying that a student cannot be enrolled in more that 5 courses
  - In ODL, we could use, instead of a set of references, an **array** of size 5
  - In E/R, we may attach a bounding number to the corresponding link



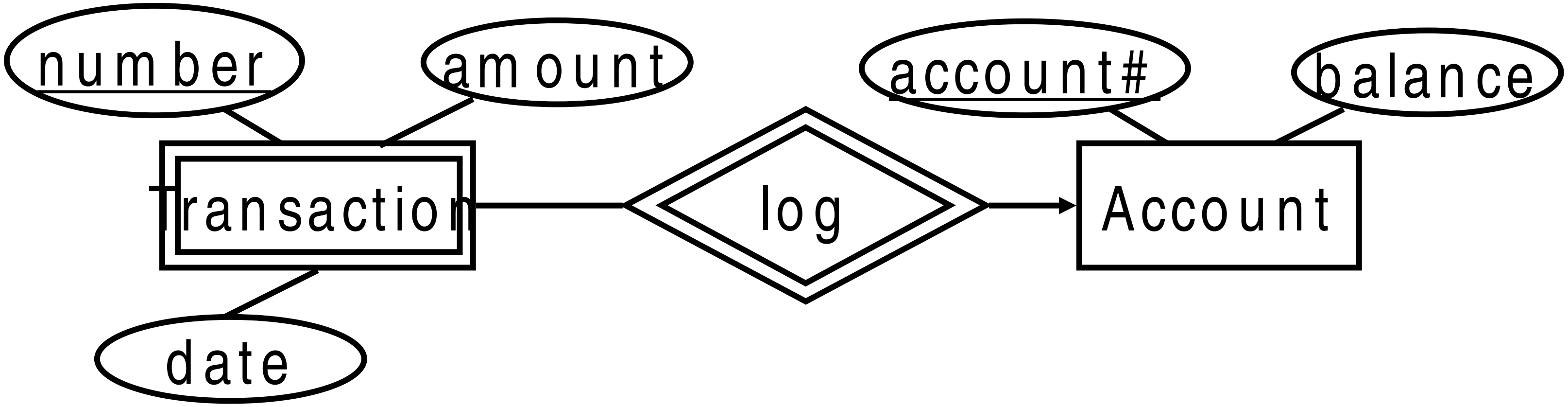
Weak Entity / Relationship Sets

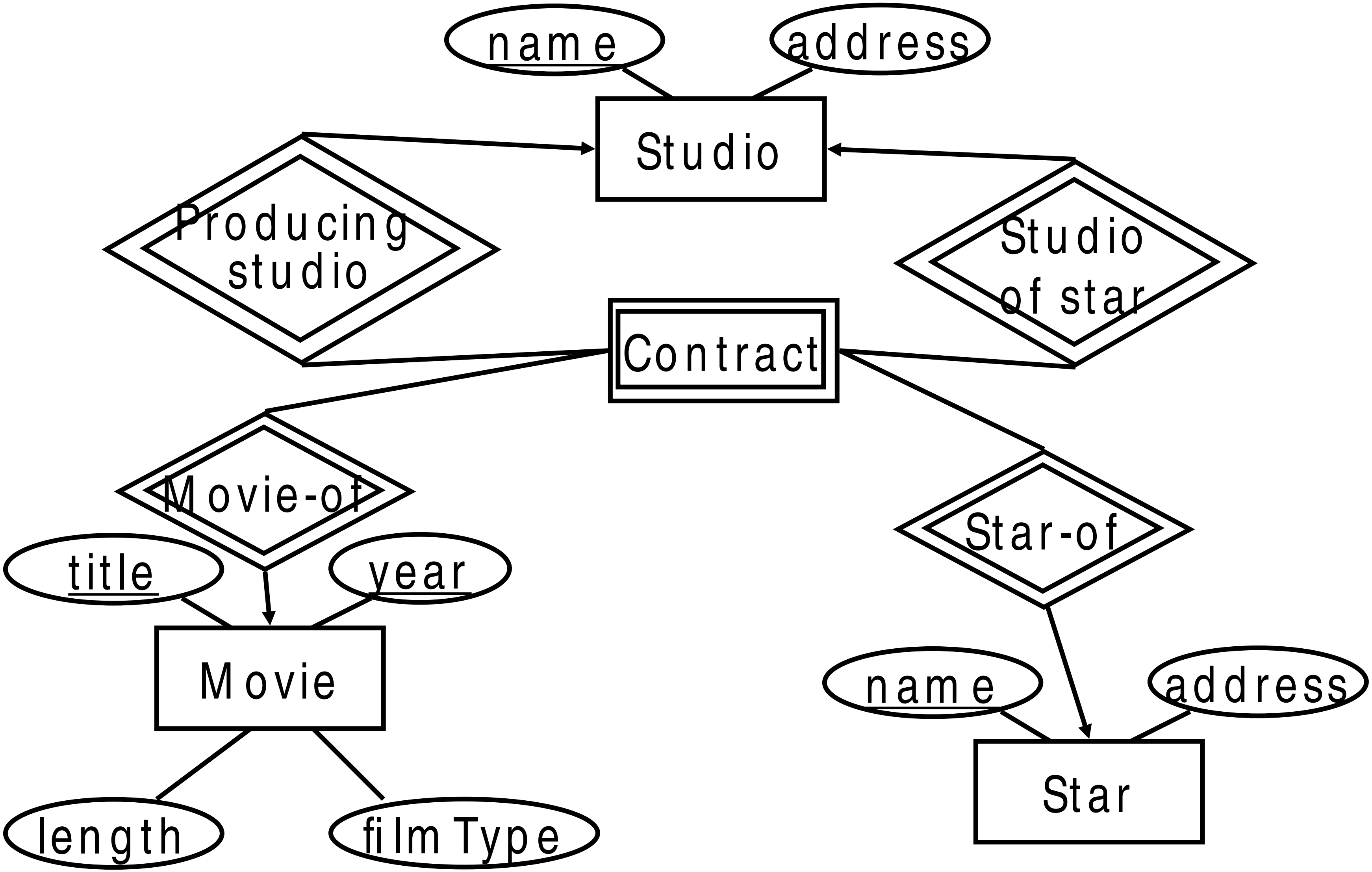
- A **strong** entity set has a primary key
- A **weak** entity set does not have sufficient attributes to form a primary key. It should be part of a one-many relationship (with no descriptive attributes) with a strong entity set
- **Discriminator** of a weak entity set is a set of attributes that distinguishes among the entities corresponding to a strong entity
- **Primary key** of a weak entity set = primary key of the strong entity + discriminator of the weak entity
- Represented in E/R model by



Example

- Log records transactions done by an ATM
- Each transaction has a number, date, and an amount
- Different accounts might have transactions by the same number, on the same date, and for the same amount




name
address
Studio
Producing studio
Studio of star
Contract
Movie-of
Star-of
title
year
Movie
length
film Type
name
address
Star

Design Principles

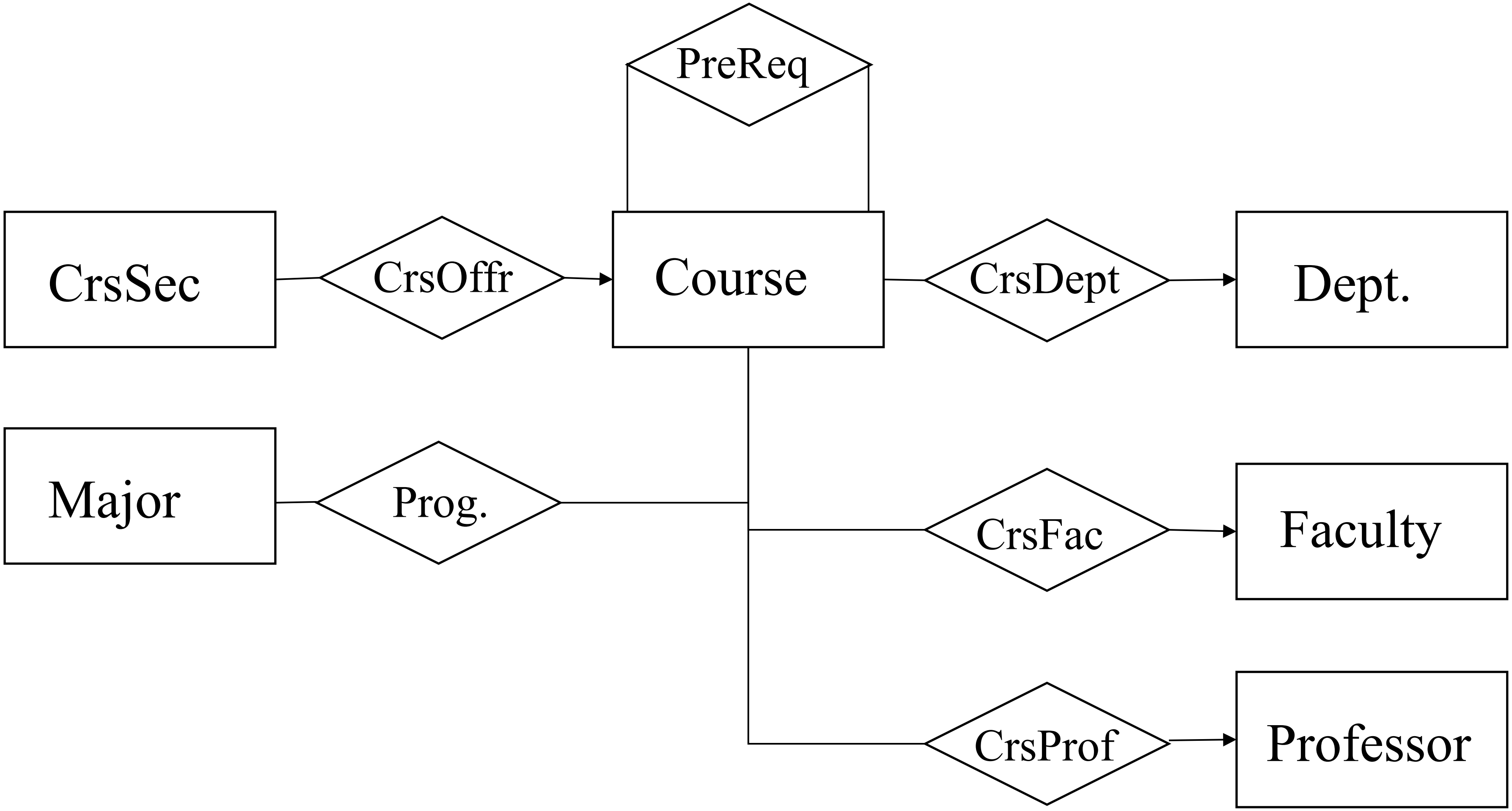
- Design should
  - Reflect reality
  - Avoid redundancy
    - Redundant information takes space
    - Could cause inconsistency
  - Be as simple as possible
- Be careful when choosing between using attributes and using classes or entity sets. Remember that
  - An attribute is **simpler** to implement than either a class/entity set or a relationship
  - If something has **more information** associated with it **than just its name**, it probably **needs to be an entity set or a class**

Consider the entity set course in a typical university :

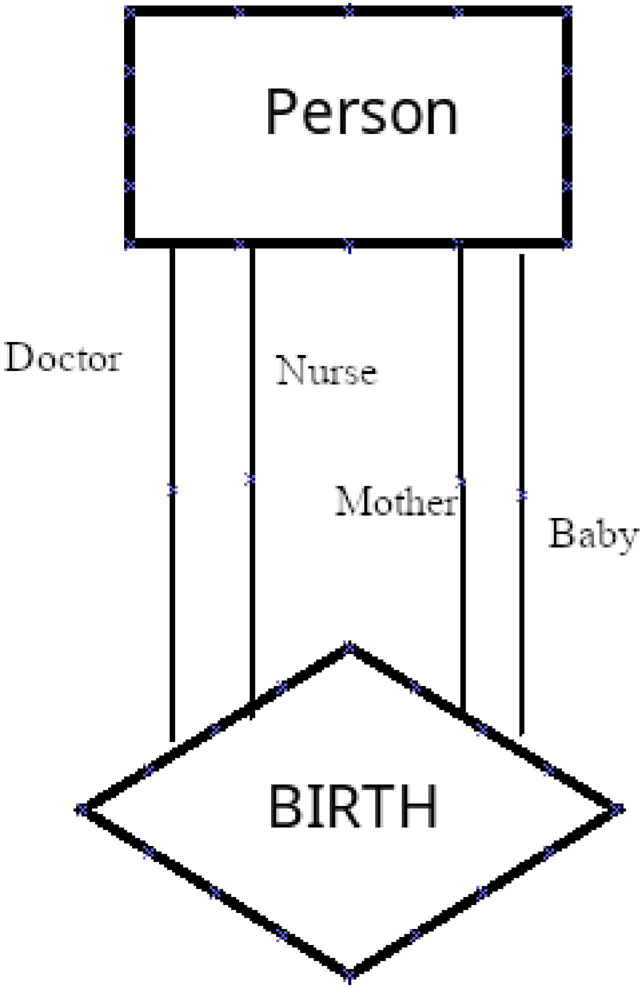
It could be involved in many relationships:
one to many relationships with
 the offering department,
 the offerings faculty
 the professor coordinating the course

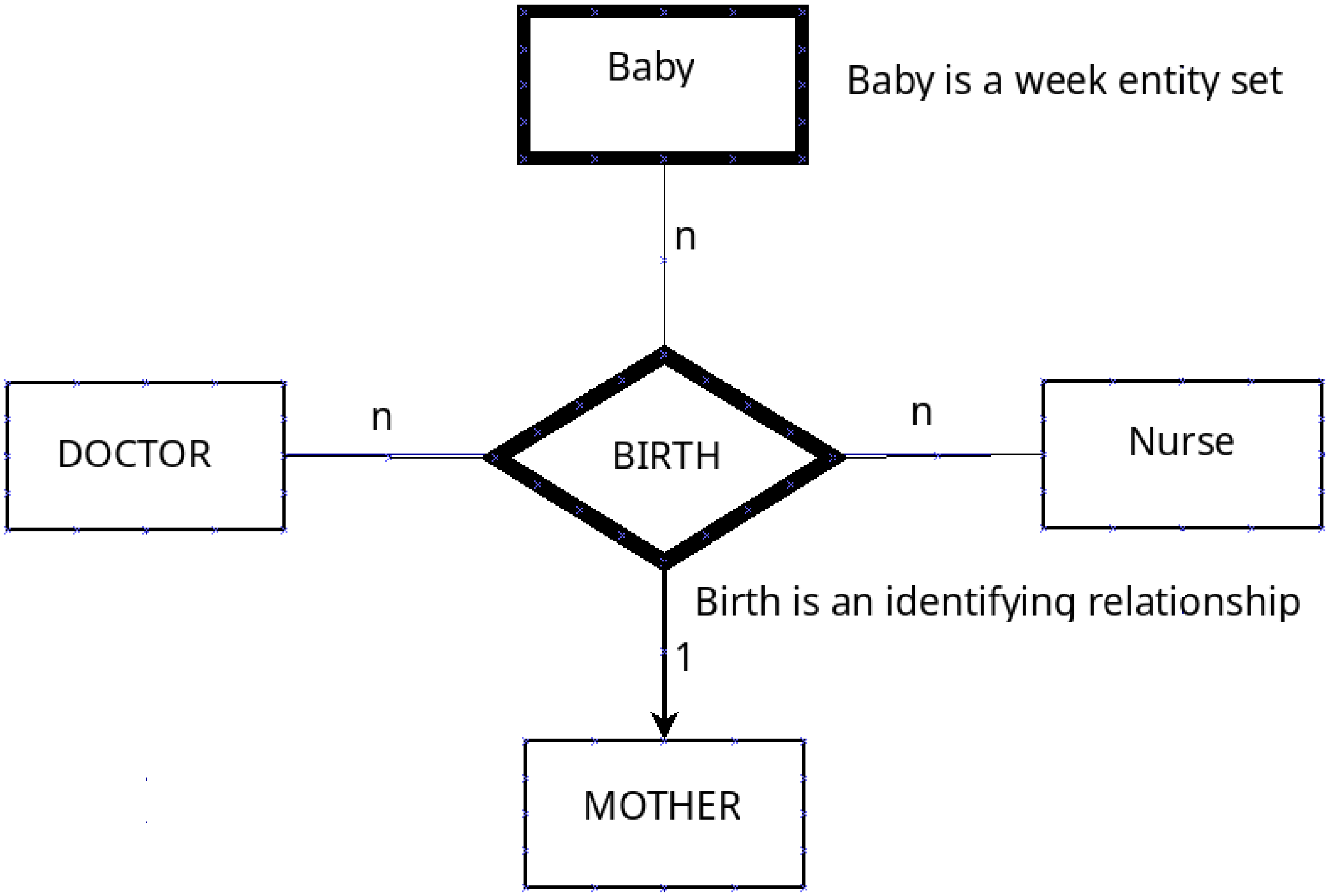
A many to one relationship with
 sections for the course

many to many relationships with
 the major program in which it is required
 the pre-requisites (follow up) courses



How to implement the entity Course and its relationship


Person
Doctor
Nurse
Mother
Baby
BIRTH


Baby
Baby is a week entity set
n
DOCTOR
n
BIRTH
n
Nurse
Birth is an identifying relationship
1
MOTHER

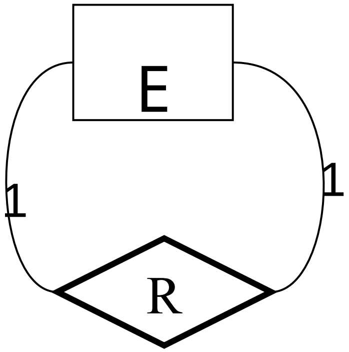
Many doctors, nurses, twins-triplets-…, one mother

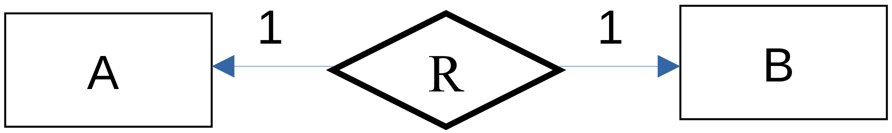
Birth(Doctor, Nurse, Baby, Mother, Time, Date, Weight)

Many doctors, nurses, twins-triplets-…, one mother

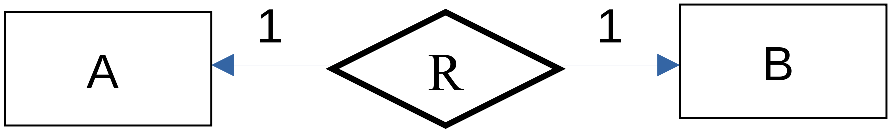
Birth(Mother, Baby, Time, Date, Weight, Doctor, Nurse)

M1	B1	T1	D1	W1	D1 D2	N1 N2 N3
M1	B2	T2	D1	W2	D1 D2	N1 N2 N3
M1	B3	T3	D1	W3	D1 D2 D3	N1 N2 N3





```
Create table R_AB(A char(2) primary key, B char(2) unique, C integer (5));
insert into R_AB values ('A1','B1',11);
Query OK, 1 row affected (0.010 sec)
insert into R_AB values ('A2','B1',11);
ERROR 1062 (23000): Duplicate entry 'B1' for key 'B'
insert into R_AB values ('A1','B2',11);
ERROR 1062 (23000): Duplicate entry 'A1' for key 'PRIMARY'
insert into R_AB values ('A2','B2',22);
Query OK, 1 row affected (0.003 sec)
select * from R_AB;
+----+------+-------+
| A  | B    | C     |
+----+------+-------+
| A1 | B1   |    11 |
| A2 | B2   |    22 |
+----+------+-------+
2 rows in set (0.000 sec)
```



Create table S_AB(A char(2), B char(2), primary key(A,B),, C integer (5));
insert into S_AB values ('A1','B1',11),('A1','B2',12), ('A2','B2',22);
Query OK, 3 rows affected (0.009 sec)
Records: 3 Duplicates: 0 Warnings: 0

select * from S_AB;

```
+----+----+-------+
| A  | B  | C     |
+----+----+-------+
| A1 | B1 |    11 |
| A1 | B2 |    12 |
| A2 | B2 |    22 |
+----+----+-------+
3 rows in set (0.000 sec)
```

Design decision

Merge the one-to-many relationships

CrsDept, CrsFac, CrsProf

in the schema for Course; all attributes of these relationships are also included in the schema for Course

Create a separate relation for each one-to-many relationships

In this case the relation for Course would have a higher arity; but requires one less join to get details for the department, faculty or professor for a given course

Similarly, merge the one-to-many relationship

CrsSec

in the schema for CrsSec

Create a relation for the many to many relationships

Program and PreReq

Course Notes

Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

Relational Database

Relational Algebra – SQL

Bipin C. DESAI

Attributes and Domains

An object or entity is characterized by its properties (attributes or data elements). The set of allowable values for an attribute is the domain of the attribute.

Domain. We define a domain, D_i, as a set of values of the same data type.
Each Attribute is defined on some underlying domain; more than one attribute may share a domain.

Tuples, Relations and Their Schemes

A relation consists of a homogeneous set of tuples.

Since each tuple in a relation represents an identifiable instance of an entity (object type), duplicate tuples are not allowed.

The number of attributes in the relation gives the **degree** or **arity** of the relation.

The **cardinality** of an instance of a relation, at a point in time, is derived from the count of the tuples in the instance.

The cardinality could change over time

Relation Representation

APPLICANT:

Name	**Age**	**Profession**
John Doe	55	Analyst
Mirian Taylor	31	Programmer
Abe Malcolm	28	Receptionist
Adrian Cook	33	Programmer
Liz Smith	33	Manager

Key. A subset of attributes X of a relation R(**R**), X ∈ **R**, with the following time independent properties is called the **key** of the relation:

Unique Identification: The values of X uniquely identify a tuple. $s[X] = t[X] \Rightarrow s = t$.

Non-redundancy: No proper subset of X has the unique identification property.

There may be more than one key in a relation; all such keys are known as **candidate keys**.

One of the candidate keys is chosen as the **primary key**; the others are known as alternate keys.
An attribute that forms part of a candidate key of a relation is called a **prime attribute**.

EMPLOYEE (***Emp#***, *Emp_Name, Profession*)
PRODUCT (***Prod#***, *Prod_Name, Prod_Details*)
JOB_FUNCTION (***Job#***, *Title*)
ASSIGNMENT (***Emp#***, ***Prod#***, ***Job#***)

EMPLOYEE

Emp#	Name	Profession
101	Jones	Analyst
103	Smith	Programmer
104	Lalonde	Receptionist
106	Letitia	VP Marketing
107	Evan	VP R & D
110	Drew	VP Operation
112	Smith	Manager

PRODUCT

Prod#	Prod_Name	Prod_Details
HEAP1	HEAP_SORT	ISS module
BINS9	BINARY_SEARCH	ISS/ R module
FM6	FILE_MANAGER	ISS/ R-PC subsys
B++1	B++_TREE	ISS/ R turbo sys
B++2	B++_TREE	ISS/ R-PC turbo

JOB_FUNCTION

Job#	Title
1000	CEO
700	Chief Programmer
800	Manager
600	Analyst

ASSIGNMENT

Emp#	Prod#	Job#
107	HEAP1	800
101	HEAP1	600
110	BINS9	800
103	HEAP1	700
101	BINS9	700
110	FM6	800
107	B++1	800

The attributes ***Emp#***, ***Prod#***, and ***Job#*** in the relation ASSIGNMENT are known as **foreign** keys.

A null value for an attribute:

- a value that is either not known at the time, or
- the value is known but not recorded, or
- no value is applicable for some tuples

P:

Id	Name
101	Jones
@	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
@	Smith

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Emp#	Name	Manager
101	Jones	@
103	Smith	110
104	Lalonde	107
107	Evan	110
110	Drew	112
112	Smith	112

Integrity rule 1 (Entity Integrity). If attribute *A* of relation R(**R**) is a component of the primary key of R(**R**), then *A* cannot accept null values.

Integrity Rule 2 (**Referential Integrity**). Given two relations R and S. Suppose R refers the relation S via a set of attribute which forms the primary key of S and, hence, this set of attributes forms a foreign key in R. Then, the value of the foreign key in a tuple in R must either be equal to the primary key of a tuple of S or be entirely null.

- All tuples which contain references to the deleted tuple should also be deleted. **cascading** deletion

- A tuple which is referred by other tuples in the database cannot be deleted.

- In the third option, the tuple is deleted, however, the foreign key attributes of all referencing tuples are set to null(otherwise "dangling" pointers!)

- All tuples which contain references to the deleted tuple should also be deleted. This is **cascading** deletion

- A tuple which is referred by other tuples in the database cannot be deleted.

- In the third option, the tuple is deleted, however, the foreign key attributes of all referencing tuples are set to null (otherwise “dangling” pointers!)

Query Languages

- The Relational model supports simple, powerful query languages which:
  - Have formal foundation based on logic.
  - Allows for implementation which can be optimized.
- Allow data access and modification.
- These languages **are not general purpose** programming languages, however, most DBMS vendors have added their own enhancements to improve its functionality.

Relational Algebra, Calculus

Relational Algebra(RA) and **Relational Calculus(RC)** are the foundation for implemented languages (e.g. SQL)

- RA is operational, and is useful for representing the plan of execution of a query.
- RC is declarative allowing users to describe what they want. (i.e. it is non-operational)

- **Understanding RA & RC is vital to the understanding of SQL and query processing!**

Your textbook may not cover RC!

Relational algebra is a collection of operations to manipulate relations.

Basic Operations: Three of these four basic operations - **union**, **intersection** and **difference** - require that operand relations be **union-compatible**. (Same number (and order)of attributes on identical (at least compatible) domains

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

UNION (∪) If we assume that P(**P**) and Q(**Q**) are two union-compatible relations, then:

The union of P(**P**) and Q(**Q**) is the set-theoretic union of P(**P**) and Q(**Q**).

The resultant relation, R = P ∪ Q, has tuples drawn from P and Q, such that

$$R = \{t \mid t \in P \lor t \in Q\} \text{ and}$$

$$\max(P, Q) \le R \le P + Q$$

Union operation is associative and commutative

$$P \cup Q \cup S = P \cup (Q \cup S) = (P \cup Q) \cup S = (P \cup S) \cup Q$$

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

P ∪ Q:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Q ∪ P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

DIFFERENCE (-)

The difference operation removes common tuples from the first relation.

$R = P - Q$ such that

$R = \{\, t \mid t \in P \;\&\; t \notin Q \}$ and $0 \leq R \leq P$

Difference operation is non-associative and non-commutative.

Is $P - Q = Q - P$?

Is $P - (Q - S) = (P - Q) - S$?

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

P - Q:

Id	Name
103	Smith
104	Lalonde
112	Smith

Q - P:

Id	Name

INTERSECTION ($\cap$)

The intersection operation selects the common tuples from the two relations.

$R = P \cap Q$ where

$R = \{ t \mid t \in P \ \& \ t \in Q\}$ and $0 \leq R \leq \min(P , Q)$

The intersection operation is really unnecessary as it can be very simply expressed as:

$$P \cap Q = P - (P - Q)$$

$$Q \cap P = Q - (Q - P)$$

Is $P – (P – Q) = Q – (Q – P)$?

P:

Id	Name
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P ∩ Q:

Id	Name
105	Letitia
107	Evan
110	Drew

Q ∩ P:

Id	Name
105	Letitia
107	Evan
110	Drew

P:

Id	Name
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

P - Q:

Id	Name
103	Smith
104	Lalonde
112	Smith

P-(P- Q)

Id	Name
105	Letitia
107	Evan
110	Drew

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

Q – P :

Id	Name
101	Jones

Q-(Q- P)

Id	Name
105	Letitia
107	Evan
110	Drew

RENAMING (ρ)

The renaming operation ρ* is used to rename relations or its attributes. The operation:

ρ(R(modattributes), rel_exp)

takes a relation expression and the result is named R with some of the attributes, specified in the modattributes, are renamed

The format of modattributes is:

modattributes ::= <oldname → newname>|

<position → newname> <,modattributes>

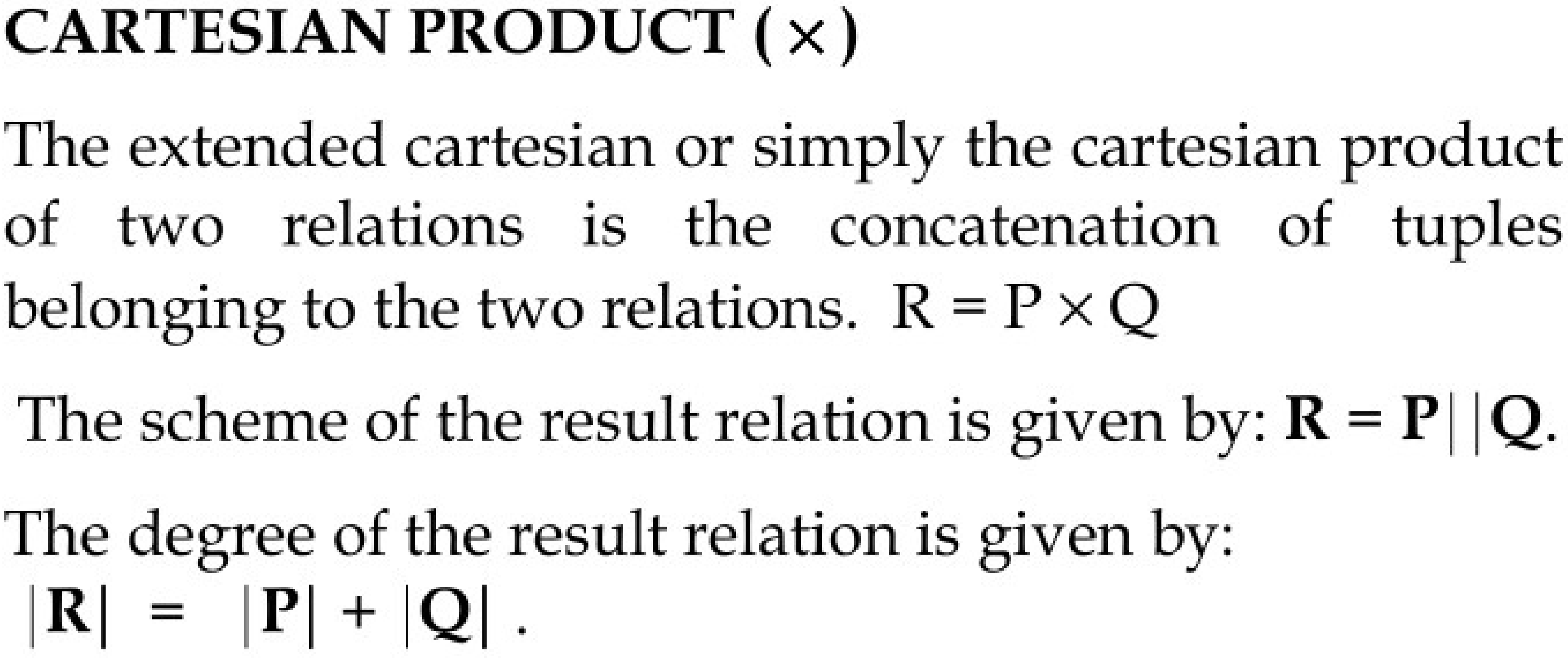
*ρ *rho* is the 17th letter of the Greek alphabet

Employee(Emp#, Ename, Address, Phone, DOB)

$$\rho(Q_{\text{Emp\#} \rightarrow \text{ID, Ename} \rightarrow \text{Name}}\ \Pi_{\text{Emp\#,Ename}}\ \text{EMPLOYEE})$$

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew



CARTESIAN PRODUCT (×)

The extended cartesian or simply the cartesian product of two relations is the concatenation of tuples belonging to the two relations. $R = P \times Q$

The scheme of the result relation is given by: $\mathbf{R} = \mathbf{P}||\mathbf{Q}$.

The degree of the result relation is given by:
$|\mathbf{R}| = |\mathbf{P}| + |\mathbf{Q}|$.

Q:

Id	Name
101	Jones
105	Letitia
107	Evan
110	Drew

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

Id	Name	Id	Name
101	Jones	101	Jones
101	Jones	103	Smith
101	Jones	104	Lalonde
…			
	…		
		…	
			….
110	Drew	112	Smith

PROJECTION ($\prod$) $\prod_X$ R

It should be noted that the projection operation reduces the arity if the number of attributes in X is less than the arity of the relation. It may also reduce the cardinality of the result relation since duplicate tuples are removed.

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

$\pi_{Name}P$:

Name
Jones
Smith
Lalonde
Letitia
Evan
Drew

SELECTION (σ)

The selection operation,yields a "horizontal subset" of a given relation. Any finite number of predicates connected by boolean operators may be specified in the selection operation.

P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia
107	Evan
110	Drew
112	Smith

$\sigma_{Id<106}$ P:

Id	Name
101	Jones
103	Smith
104	Lalonde
105	Letitia

JOIN (▯ ▯)

The join operator, as the name suggests, allows the combining of two relations to form a single new relation.

- first compute the cartesian product
- followed by selecting those tuples where the common attribute(s) has(have) the same value(s).

Project (Proj#, Pname, Pleader) Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB)

- **Get Emp# of employees working on Proj# comp353.**
- **Get complete details of employee working on comp353.**
- **Get complete detatils of employes working on the Database project.**
- **Get complete details of employees working on both comp353 and comp354**
- **Get Emp# (complete details) of employees working on two projects.**
- **Get names of employees working in projects where Ma is the project leader.**

Project (Proj#, Pname, Pleader) Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

❖**Get Emp# of employees working on Proj# comp353.**

$$\Pi_{E\#}\ (\sigma_{P\#=comp353}(Assign))$$

❖**Get complete details of employee working on comp353.**

$$Employee\ \triangleright\triangleleft_{Emp\#=E\#}\ \Pi_{E\#}\ (\sigma_{P\#=comp353}(Assign))$$

Project (Proj#, Pname, Pleader)   Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

- ❖ **Get complete details of employees working on the Database project(s) – a project name.**

$$X = \Pi_{E\#} \left((\text{Assign}) \bowtie_{\text{Proj\#=P\#}} (\sigma_{\text{Pname="Database"}}(\text{Project})) \right)$$

$$\text{Employee} \bowtie_{\text{Emp\#=E\#}} X$$

Combining in one RA expression:

$$\text{Employee} \bowtie_{\text{Emp\#=E\#}} \Pi_{E\#} \left((\text{Assign}) \bowtie_{\text{Proj\#=P\#}} (\sigma_{\text{Pname="Database"}}(\text{Project})) \right)$$

Project (Proj#, Pname, Pleader) Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

- **Get complete details of employees working on both comp353 and comp354**

$$
\begin{array}{l}
\text{Employee } \triangleright\triangleleft_{\text{Emp\#=E\#}} \Pi_{\text{E\#}} (\sigma_{\text{P\#=comp353}}(\text{Assign})) \cap \\
\text{Employee } \triangleright\triangleleft_{\text{Emp\#=E\#}} \Pi_{\text{E\#}} (\sigma_{\text{P\#=comp354}}(\text{Assign})) \\
\text{or Employee } \triangleright\triangleleft_{\text{Emp\#=E\#}} (\Pi_{\text{E\#}} (\sigma_{\text{P\#=comp353}}(\text{Assign})) \cap \\
\qquad\qquad\qquad\qquad \Pi_{\text{E\#}} (\sigma_{\text{P\#=comp354}}(\text{Assign}))
\end{array}
$$

Project (Proj#, Pname, Pleader)   Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

- **Get complete details of employees working on (any)two projects**

  $\rho(\text{A1(P1\#, E1\#)}, \text{Assign})$
  $\text{X} = \text{A1} \times \text{Assign} \qquad \text{Y} = \Pi_{\text{E\#}} (\sigma_{\text{P\#} \neq \text{P1\#} \wedge \text{E\#=E1\#}} (\text{X}))$
  $\text{Employee} \rhd\lhd_{\text{Emp\#=E\#}} \text{Y}$

Combining in one RA expression:

$$\text{Employee} \rhd\lhd_{\text{Emp\#=E\#}} (\Pi_{\text{E\#}} (\sigma_{\text{P\#} \neq \text{P1\#} \wedge \text{E\#=E1\#}} (\rho(\text{A1(P1\#, E1\#)}, \text{Assign}) \times \text{Assign})))$$

Project (Proj#, Pname, Pleader) Assign(P#, E#)

Employee(Emp#, Ename, Address, Phone, DOB,)

- Get complete details of employees working in projects where Ma(an employee name) is the project leader.

$X= (\sigma_{Ename=\text{"Ma"}}(Employee)))$

$Y=\Pi_{Proj\#} (Project \rhd\lhd_{Emp\#=Pleader} X)$

$Z=\Pi_{E\#} (Assign \rhd\lhd_{Proj\#=P\#} Y)$

$Employee \rhd\lhd_{Emp\#=E\#} Z$

$Employee \rhd\lhd_{Emp\#=E\#} (\Pi_{E\#} (Assign \rhd\lhd_{Proj\#=P\#} (\Pi_{Proj\#} (Project \rhd\lhd_{Emp\#=Pleader} (\sigma_{Ename=\text{"Ma"}}(Employee)))))))$

Complete set of RA operations

$$\{ \sigma,\ \Pi,\ \cup,\ -,\ \times \}$$

The above is a complete set of RA operations.

The others could be expressed as a sequence

of operations from this set.

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S-R))$$

$R \bowtie_B S \equiv \sigma_B (R \times S)$ etc

Definition: **Theta-join**. The **theta-join** of two relations P(**P**) and Q(**Q**) is defined as

$R = P \bowtie_B Q$

such that $R = \{t \mid t_1 || t_2 \wedge t_1 \in P \wedge t_2 \in Q \wedge B\}$

where B is a selection predicate consisting of terms of the form: $(t_1[A_i]\ \theta\ t_2[B_i])$ for i = 1, 2, ..., n,

where θ_i is some comparison operator ($\theta \in \{=,\neq,<,\leq,>,\geq\}$), and A_i and B_i are some domain compatible attributes of the relation schemes **P** and **Q** respectively.

$0 \leq |R| \leq |P| * |Q|$

$|\mathbf{R}| = |\mathbf{P}| + |\mathbf{Q}|$

Two common and very useful variants of the join are the **equi-join** and the **natural-join**.

If two relations that are to be joined have no domain compatible attributes, then the natural join operation is equivalent to a simple cartesian product.

-The equi-join and the theta joins are "horizontal subsets" of the cartesian product.

-The natural join is equivalent to an equi-join with a subsequent projection to eliminate the duplicate attributes.

SQL is both the data definition and data manipulation language of the relational database systems

Create table <relation> (<attribute list>, <integrity constraint list>)
Where the <attribute list> is specified as:
<Attribute list> ::= <attribute name> (<data type>)[not null][,<attribute list>]
and <integrity constraints list> is specified as:
<Integrity constraint list> ::= <integrity1> | <integrity constraint list>
and <integrity> could be a primary key(a1, a2, … am) , not null or a named constraint.

```
create table EMPLOYEE
    (Empl_No  integer not null,
     Name     char(25),
     Skill    char(20),
     Pay_Rate decimal(10,2)
     Primary key Empl_No )
```

```
select [distinct] <target list>
from  <relation list>
[where <predicate>]
```

Database Schema

- Employee(Name, Sin, Dept#, MGRSIN)
- Dept(Dname, Dept#, MgrSin, Bcode)
- Project(Pname, Proj#, Dept#, Lab)
- Assign(Proj#, EmpSin, Hours)
- EmpDet(Sin, Address, Salary, DOB)
- EmplDepd(Sin, DepName, HowR)

Queries

Employee(Name, Sin, Dept#, MGRSIN)
Dept(Dname, Dept#, MgrSin, Bcode)
Project(Pname, Proj#, Dept#, Lab)

- Names of Employees in Dept 101?

```
select Name
from Employee
where Dept#= 101
```

- Details of Employee in Dept. 101?

```
select *
from Employee
where Dept#= 101
```

Employee(Name, Sin, Dept#, MGRSIN)
Dept(Dname, Dept#, MgrSin, Bcode)
Project(Pname, Proj#, Dept#, Lab)

- For all projects in the Software Engg. Lab, find the DName, Manager’s name?

```
select Dname, Name
from Employee e, Dept d, Project p
where Lab = ‘Software Engg.’
        and p.Dept#=d.Dept#
        and d.MgrSin=e.Sin
```

- For all projects in the Software Engg. Lab, find the DName, Manager’s address etc.

Employee(Name, Sin, Dept#, MGRSIN)
Dept(Dname, Dept#, MgrSin, Bcode)
Project(Pname, Proj#, Dept#, Lab)
EmpDet(Sin, Address, Salary, DOB)

✯ For all projects in the Software Engg. Lab, find the DName, Manager’s name, address etc.

```
select Dname, Name,Address, Salary,DOB
from Employee e, Dept d, Project p, EmpDet t
where Lab = ‘Software Engg.’
        and p.Dept#=d.Dept#
        and d.MgrSin=e.Sin
        and e.Sin=t.Sin
```

Query Tree

Consider the relation:
Employee(Emp#, Ename, City, Phone, YOB,)

Suppose we want to find Emp# and names of employees who live in NDG(an address) and who were born in 1971(YOB).

We can express this query in one of the following ways:

$\Pi_{\text{Emp\#, Ename}} (\sigma_{\text{City='NDG'} \wedge \text{YOB =1971}}(\text{EMPLOYEE}))$

or

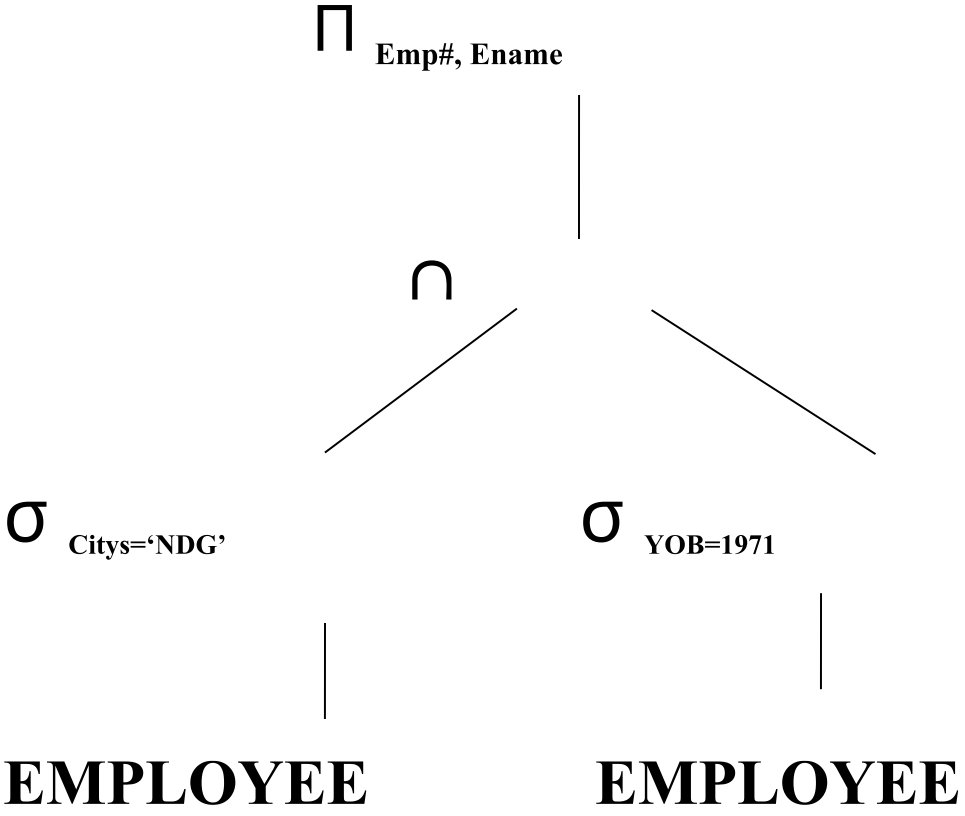
$\Pi_{\text{Emp\#, Ename}} (\sigma_{\text{City='NDG'}}(\text{EMPLOYEE}) \cap \sigma_{\text{YOB =1971}}(\text{EMPLOYEE}))$

or

$\Pi_{\text{Emp\#, Ename}} \sigma_{\text{City='NDG'}}(\text{EMPLOYEE}) \cap$

$\Pi_{\text{Emp\#, Ename}} \sigma_{\text{YOB =1971}}(\text{EMPLOYEE})$

$\Pi_{\text{Emp\#, Ename}} (\sigma_{\text{City=‘NDG’}\ {}^{\wedge}\ \text{YOB =1971}}(\text{EMPLOYEE}))$



$\Pi_{\text{Emp\#, Ename}}$

$\cap$

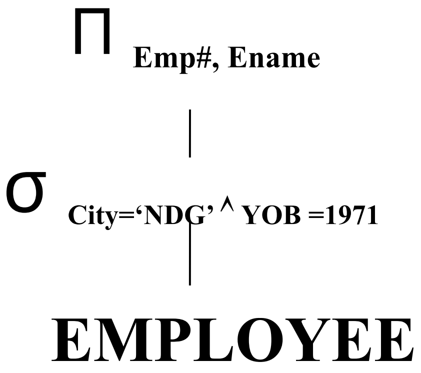
$\sigma_{\text{Citys=‘NDG’}}$

$\sigma_{\text{YOB=1971}}$

EMPLOYEE

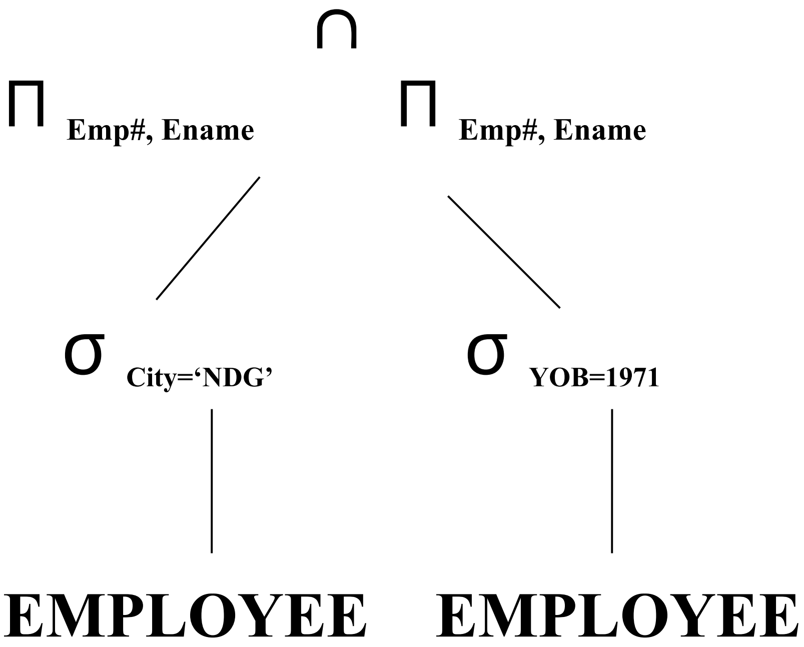
EMPLOYEE

$\Pi_{\text{Emp\#, Ename}}\,(\sigma_{\text{City='NDG'}}(\text{EMPLOYEE}) \cap \sigma_{\text{YOB =1971}}(\text{EMPLOYEE}))$



$\Pi_{\text{Emp\#, Ename}}\ \sigma_{\text{City='NDG'}}(\text{EMPLOYEE}) \cap$

$\Pi_{\text{Emp\#, Ename}}\ \sigma_{\text{YOB =1971}}(\text{EMPLOYEE})$



Project (Proj#, Pname, Pleader)	100 projects
Empl (Emp#, Ename, City, Ph, DOB,)	500 employees
Assign (P#, E#)	1500 assignments

Get complete details of employees working on the DB project(s).

Suppose there are 10 DB projects, distribution is uniform.

Av. of 3 projects for each employee; Av. of 15 employees per project

Number of assignments to DB project is 150 (10% of assignments).

If no employee works on more than one DB project,

then maximum number of tuples in output would be 150.

Two possible ways of expressing this query

$$\Pi_{Emp\#,\ Ename,\ City,\ Ph,\ DOB}(\sigma_{E\#=Emp\#}(\sigma_{P\#=Proj\#}(\sigma_{Pname=\text{“DB”}}(Empl \times Assign \times Project))))$$

$$Empl \bowtie_{Emp\#=E\#} (\Pi_{E\#} ((Assign) \bowtie_{Proj\#=P\#} (\sigma_{Pname=\text{“DB”}}(Project))))$$

Pr	Pn
P1	DB
P2	X

P	E
P1	E1
P2	E2

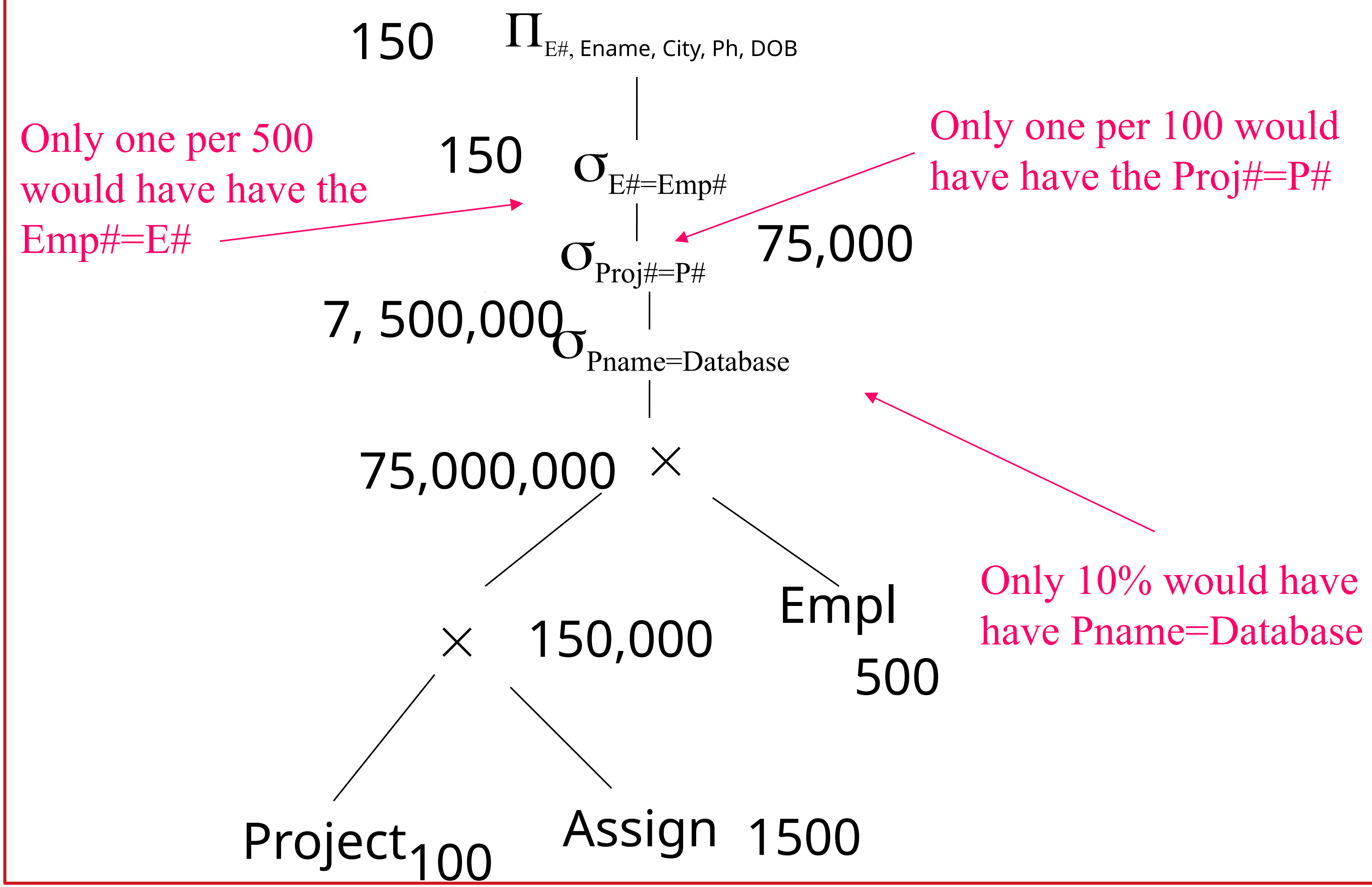
Em	En
E1	N1
E2	N2

. . .

. . .

Pr	Pn	P	E	Em	En
P1	DB	P1	E1	E1	N1
				E2	N2
		....			
		P2	E2	E1	N1
				E2	N2
		....			
P2	X	P1	E1	E1	N1
				E2	N2
		....			
		P2	E2	E1	N1
				E2	N2
		....			

$\Pi_{E\#,\ Ename,\ City,\ Ph,\ DOB}(\sigma_{P\#=Database}\ (Empl \times Assign \times Project))$

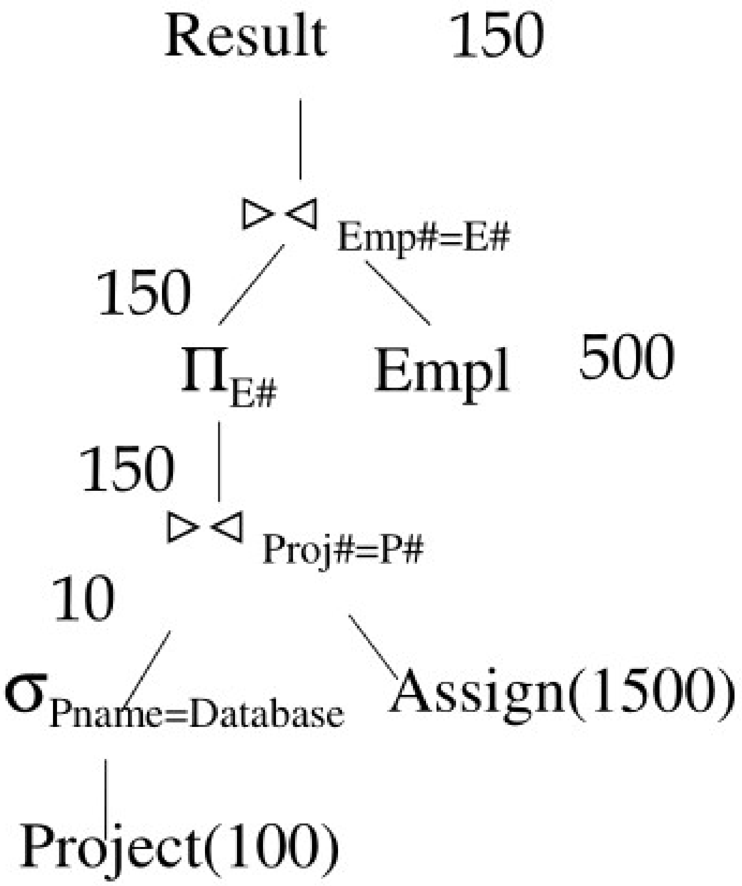


Only one per 500 would have have the Emp#=E#

Only one per 100 would have have the Proj#=P#

Only 10% would have have Pname=Database

$$Empl \;\triangleright\triangleleft_{Emp\#=E\#} (\Pi_{E\#} ((Assign) \triangleright\triangleleft_{Proj\#=P\#} (\sigma_{P\#=Database}(Project))))$$



Division (÷)

P(**P**):

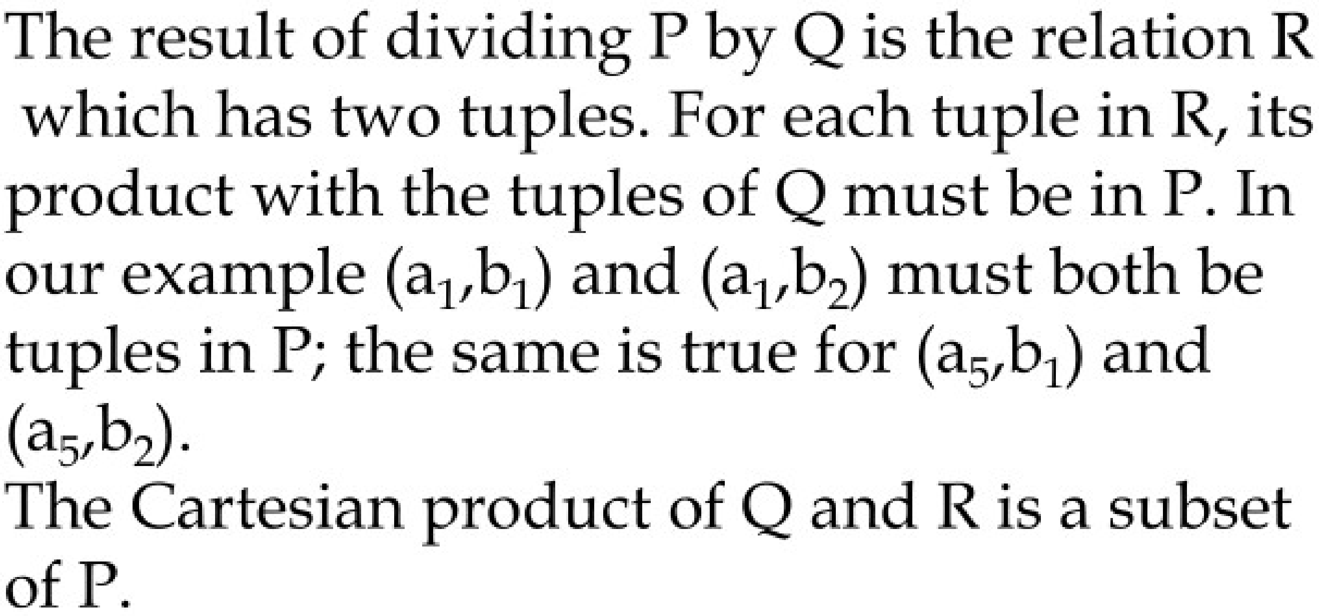
A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_2
a_5	b_1
a_5	b_2

Q(**Q**):

B
b_1
b_2

R(R)(result):

A
a_1
a_5



The result of dividing P by Q is the relation R which has two tuples. For each tuple in R, its product with the tuples of Q must be in P. In our example (a_1,b_1) and (a_1,b_2) must both be tuples in P; the same is true for (a_5,b_1) and (a_5,b_2).

The Cartesian product of Q and R is a subset of P.

P(**P**):

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_2
a_5	b_1
a_5	b_2

Q(**Q**):

B
b_1

R(**R**) is:

A
a_1
a_2
a_3
a_5

Q(**Q**):

B
b_1
b_2
b_3

R(R) is:

A

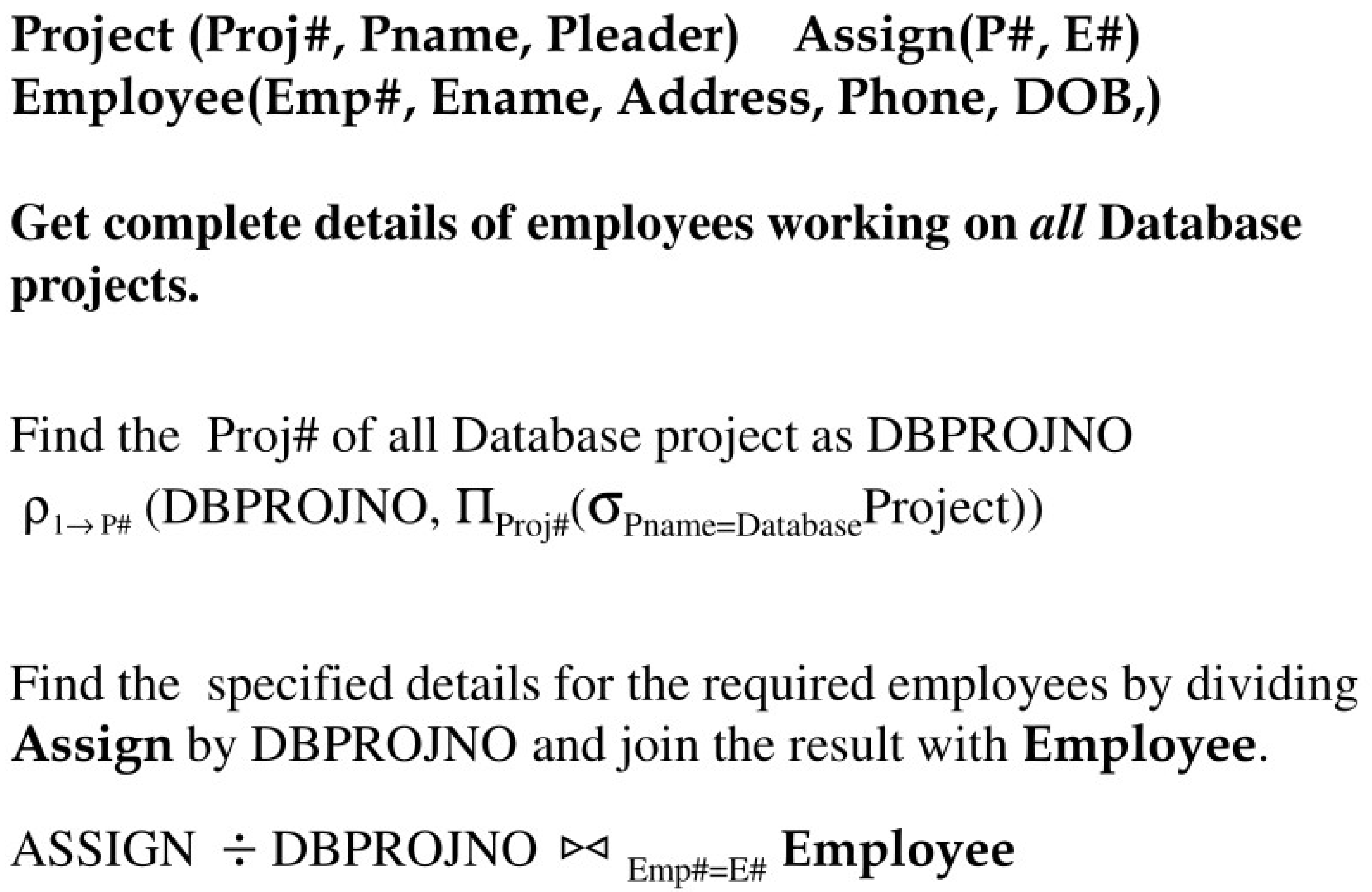
Q(**Q**):

B

R(R) is:

A
a_1
a_2
a_3
a_4
a_5

The division operation is useful where a query involves the phrase "*for all objects having all of the specified properties*" .



Project (Proj#, Pname, Pleader) **Assign(P#, E#)**
Employee(Emp#, Ename, Address, Phone, DOB,)

Get complete details of employees working on *all* Database projects.

Find the Proj# of all Database project as DBPROJNO

$$\rho_{1\rightarrow P\#}(\mathrm{DBPROJNO}, \Pi_{Proj\#}(\sigma_{Pname=Database}\mathrm{Project}))$$

Find the specified details for the required employees by dividing **Assign** by DBPROJNO and join the result with **Employee**.

$$\mathrm{ASSIGN} \div \mathrm{DBPROJNO} \bowtie_{Emp\#=E\#} \mathbf{Employee}$$

Project (Proj#, Pname, Pleader)   Assign(P#, E#)
Employee(Emp#, Ename, Address, Phone, DOB,)

Get complete details of employees working exactly on all DB projects.

Find the Proj# of all Database project as DBPROJNO

$\rho_{1 \rightarrow P\#}$ (DBPROJNO, $\Pi_{Proj\#}(\sigma_{Pname=Database}$Project))

Find those employees who work on all DB projects by dividing **Assign** by DBPROJNO (some of them work on other projects as well!).

ALLDB =ASSIGN $\div$ DBPROJNO

Find those tuples not involving assignments to DB projects

NOTDBONLY =ASSIGN – DBPROJNO × ALLDB

Required employees: ONLYDB = ALLDB - $\Pi_{E\#}$ NOTDBONLY

Result is: ONLYDB $\bowtie_{Emp\#=E\#}$ **Employee**

Division is not a basic operation

We can re-write **P(AB)÷Q(B)** by :

$$\Pi_A P - \Pi_A(\Pi_A P \times Q - P)$$

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_3	b_1
a_4	b_2
a_5	b_1
a_5	b_2

B
b_1
b_2

$\Pi_A P$
a_1
a_2
a_3
a_4
a_5

$\Pi_A P \times Q$

A	B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_2
a_3	b_1
a_3	b_2
a_4	b_1
a_4	b_2
a_5	b_1
a_5	b_2

$\Pi_A P \times Q - P$

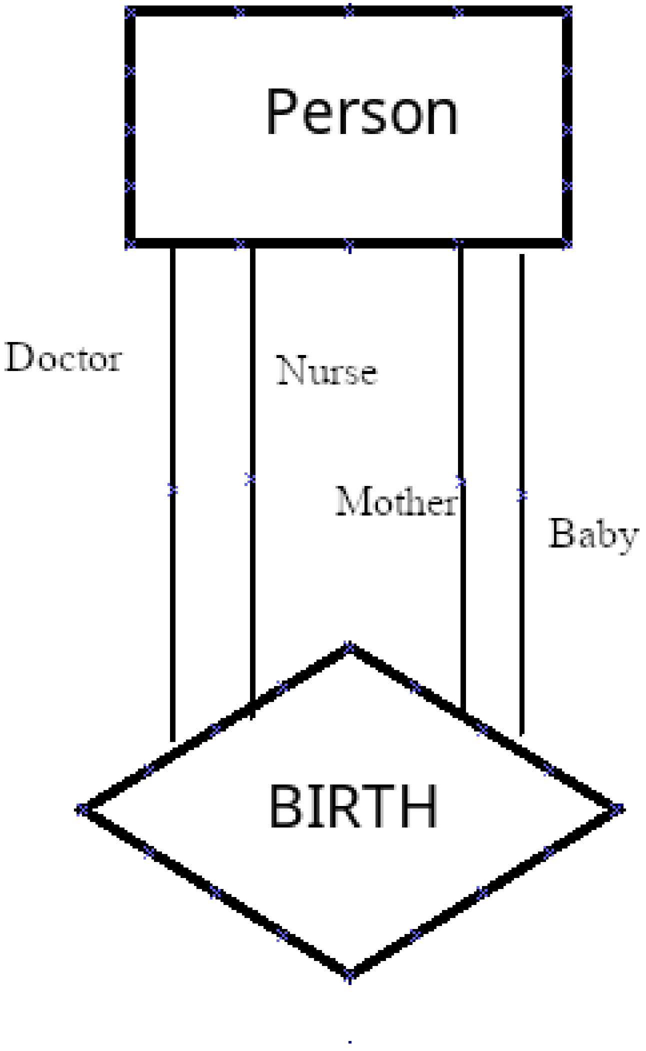
A	B
a_2	b_2
a_3	b_2
a_4	b_1

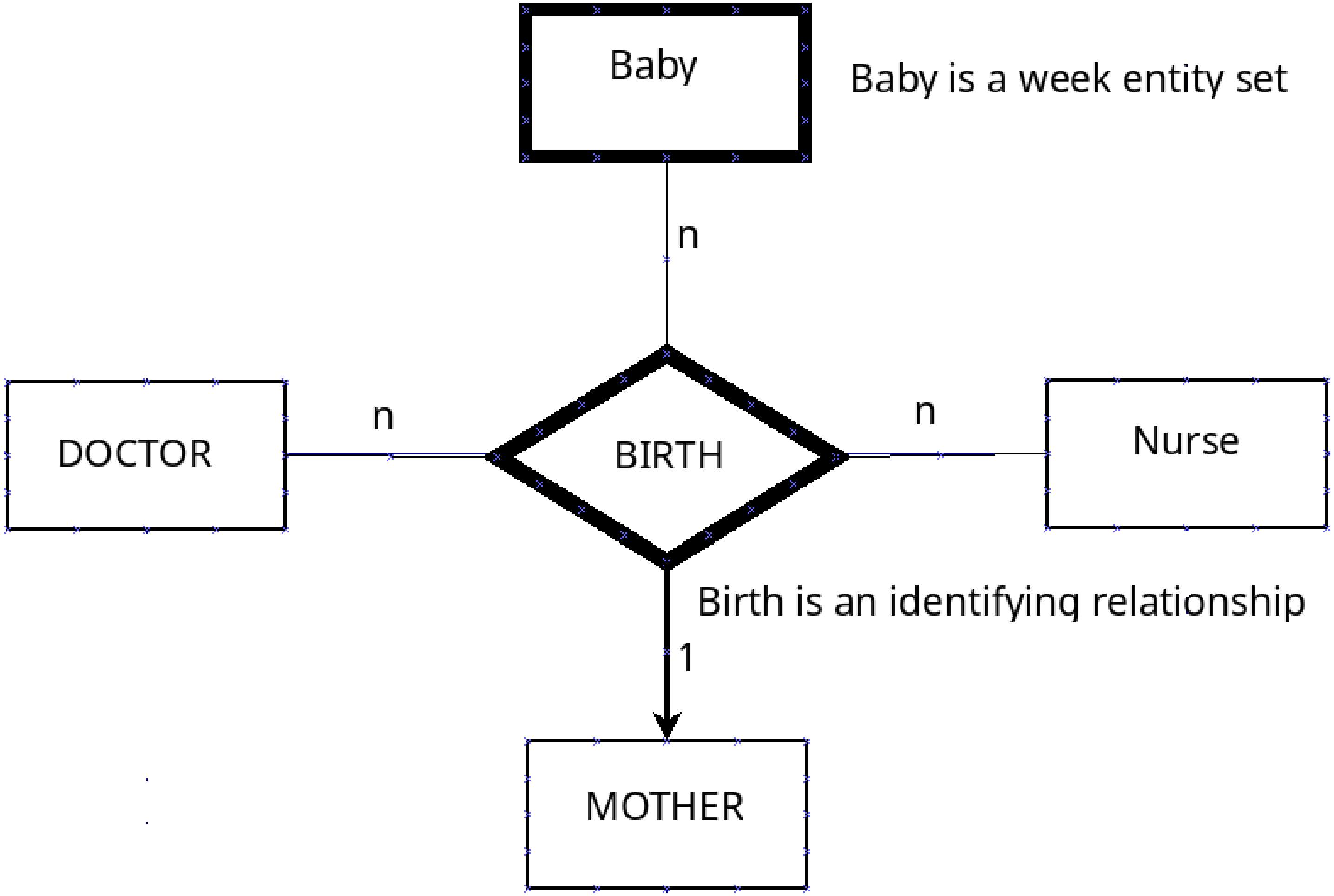
$\Pi_A(\Pi_A P \times Q - P)$

A
a_2
a_3
a_4

$\Pi_A P - \Pi_A(\Pi_A P \times Q - P)$

A
a_1
a_5


Person
Doctor
Nurse
Mother
Baby
BIRTH


Baby
Baby is a week entity set
n
DOCTOR
n
BIRTH
n
Nurse
Birth is an identifying relationship
1
MOTHER

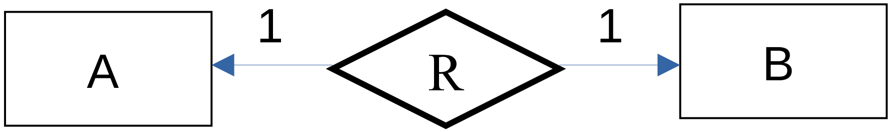
Many doctors, nurses, twins-triplets-…, one mother

Birth(Doctor, Nurse, Baby, Mother, Time, Date, Weight)

Many doctors, nurses, twins-triplets-…, one mother

Birth(Mother, Baby, Time, Date, Weight, Doctor, Nurse)

M1	B1	T1	D1	W1	D1 D2	N1 N2 N3
M1	B2	T2	D1	W2	D1 D2	N1 N2 N3
M1	B3	T3	D1	W3	D1 D2 D3	N1 N2 N3

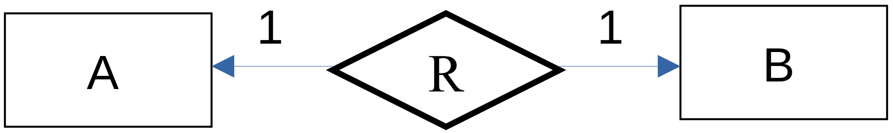


Create table R_AB(A char(2), B char(2), primary key(A,B),, C integer (5));
insert into R_AB values ('A1','B1',11),('A1','B2',12), ('A2','B2',22);
Query OK, 3 rows affected (0.009 sec)
Records: 3 Duplicates: 0 Warnings: 0

select * from R_AB;

```
+----+----+------+
| A  | B  | C    |
+----+----+------+
| A1 | B1 |   11 |
| A1 | B2 |   12 |
| A2 | B2 |   22 |
+----+----+------+
3 rows in set (0.000 sec)
```

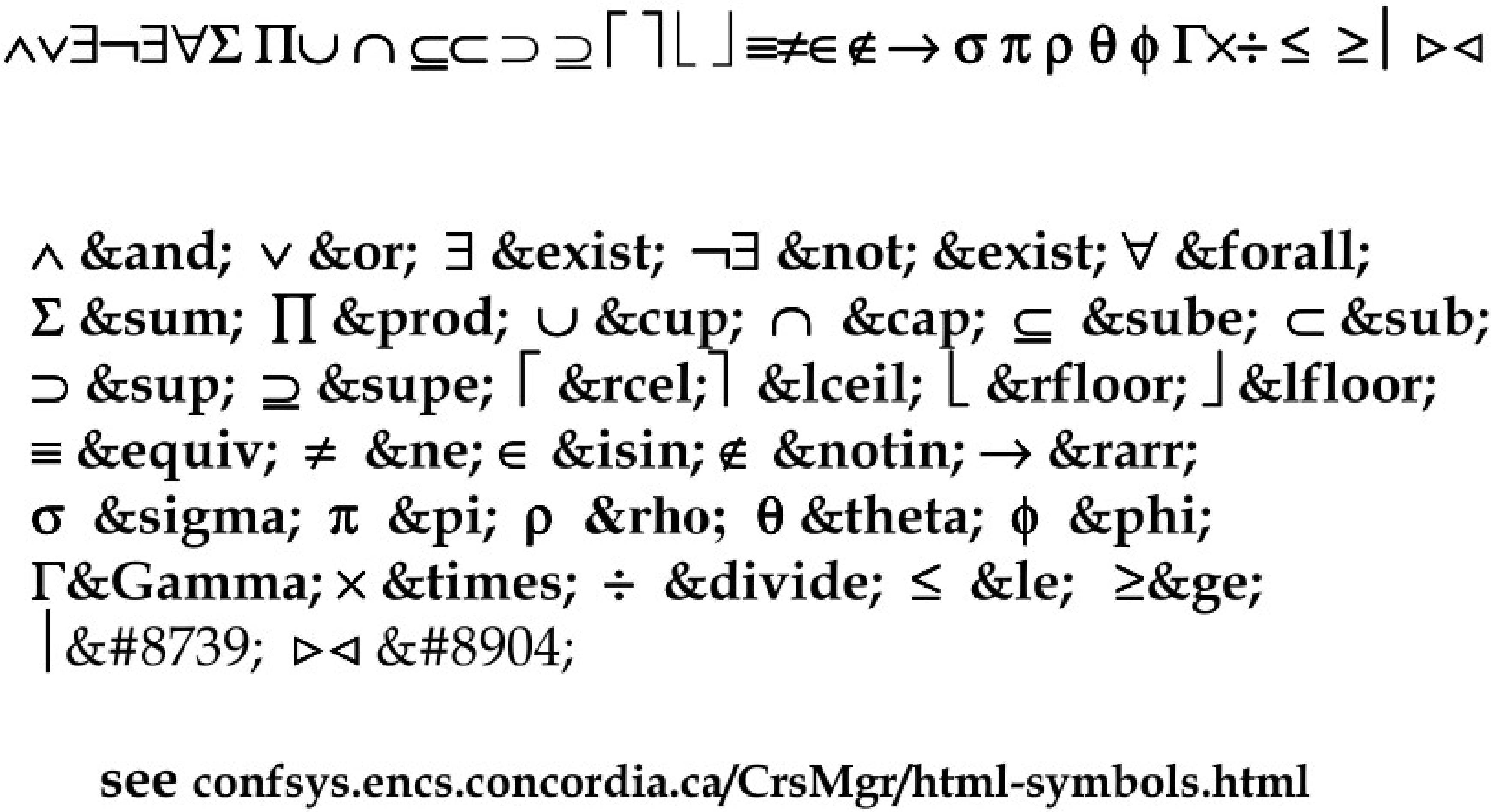
NOT 1:1



```
Create table R_AB(A char(2) primary key, B char(2) unique, C integer (5));
insert into R_AB values ('A1','B1',11);
Query OK, 1 row affected (0.010 sec)
insert into R_AB values ('A2','B1',11);
ERROR 1062 (23000): Duplicate entry 'B1' for key 'B'
insert into R_AB values ('A1','B2',11);
ERROR 1062 (23000): Duplicate entry 'A1' for key 'PRIMARY'
insert into R_AB values ('A2','B2',22);
Query OK, 1 row affected (0.003 sec)
select * from R_AB;
+----+------+------+
| A  | B    | C    |
+----+------+------+
| A1 | B1   |   11 |
| A2 | B2   |   22 |
+----+------+------+
2 rows in set (0.000 sec)
```

1:1

Some special characters used in DB & HTML codes



∧∨∃¬∃∀Σ∏∪∩⊆⊂⊃⊇⌈⌉⌊⌋≡≠∈∉→σπρθϕΓ×÷≤ ≥∣ ▷◁

∧ &and; ∨ &or; ∃ &exist; ¬∃ &not; &exist; ∀ &forall;
Σ &sum; ∏ &prod; ∪ &cup; ∩ &cap; ⊆ &sube; ⊂ &sub;
⊃ &sup; ⊇ &supe; ⌈ &rcel;⌉ &lceil; ⌊ &rfloor; ⌋ &lfloor;
≡ &equiv; ≠ &ne; ∈ &isin; ∉ &notin; → &rarr;
σ &sigma; π &pi; ρ &rho; θ &theta; ϕ &phi;
Γ&Gamma; × &times; ÷ &divide; ≤ &le; ≥&ge;
∣ &#8739; ▷◁ &#8904;

see confsys.encs.concordia.ca/CrsMgr/html-symbols.html

Course Notes
Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

This document in electronic form, bearing a CopyForward permission, could be used for personal use and/or study, free of charge. Anyone could use it to derive updated versions. The derived version must be published under CopyForward. All authors of the version used to derive the new version must be included in the updated version in the existing order, followed by name(s) of author(s) producing the derived work

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

Relational Model

&

Relational Database Design

Bipin C. DESAI

- Relation Scheme and Relational Design

- Anomalies in Database:
- A Consequence of Bad Design

- Functional Dependency

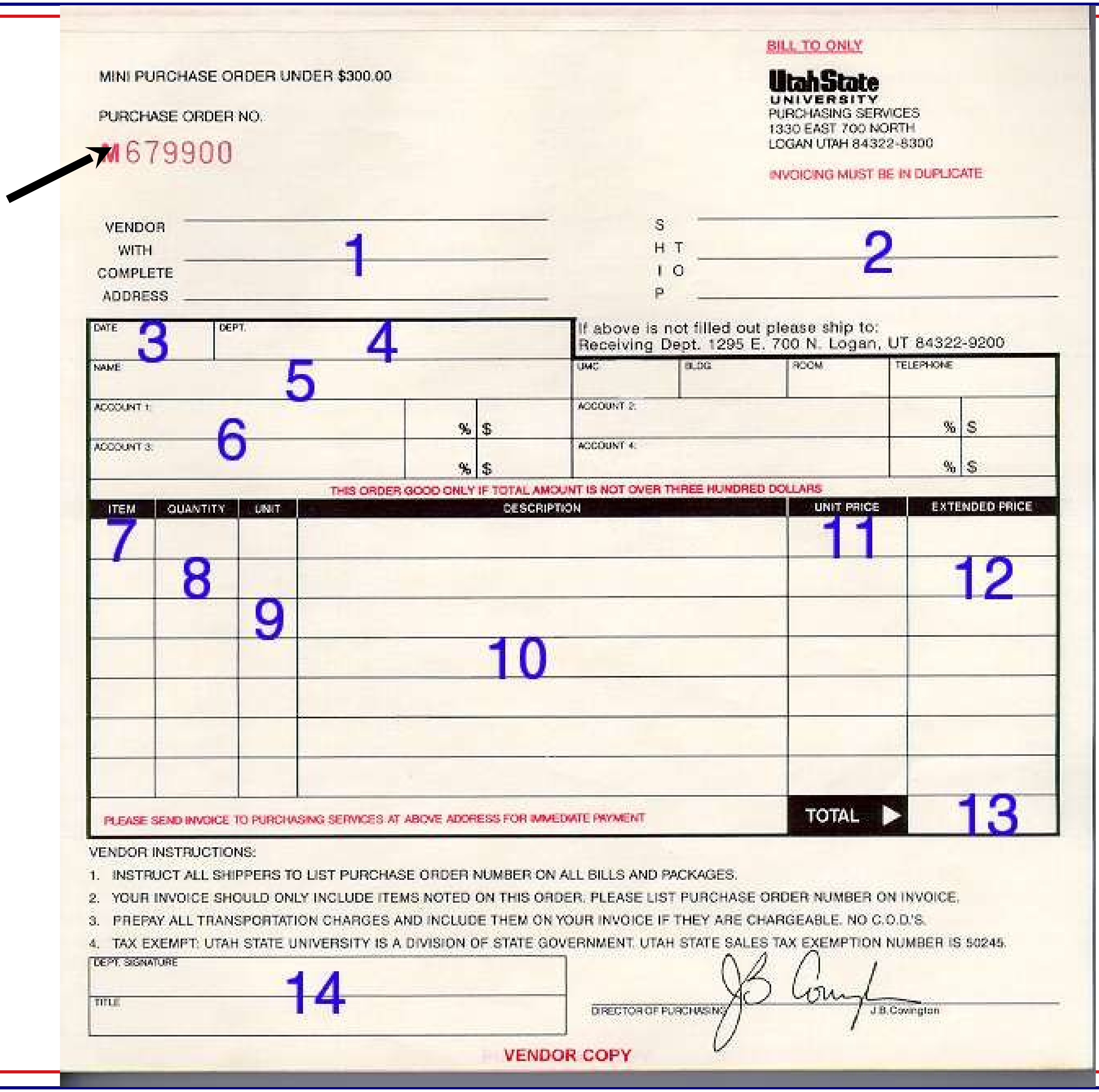
- Normal Forms

- A relation scheme **R** is a plan which gives the attributes involved in one or more relations defined on this relation scheme.

$\{A_1, A_2, \ldots, A_n\}$,

A_i is defined on domain $\mathbf{D_i}$

Relation R on the relation scheme **R** is a finite set of mappings or tuples $\{t_1, t_2, \ldots, t_p\}$



MINI PURCHASE ORDER UNDER $300.00

PURCHASE ORDER NO.

M679900

BILL TO ONLY

UtahState
UNIVERSITY
PURCHASING SERVICES
1330 EAST 700 NORTH
LOGAN UTAH 84322-8300

INVOICING MUST BE IN DUPLICATE

VENDOR
WITH
COMPLETE
ADDRESS

SHIP TO

If above is not filled out please ship to:
Receiving Dept. 1295 E. 700 N. Logan, UT 84322-9200

DATE	DEPT.					
NAME			UMC	BLDG	ROOM	TELEPHONE
ACCOUNT 1:	%	$	ACCOUNT 2:		%	$
ACCOUNT 3:	%	$	ACCOUNT 4:		%	$

THIS ORDER GOOD ONLY IF TOTAL AMOUNT IS NOT OVER THREE HUNDRED DOLLARS

ITEM	QUANTITY	UNIT	DESCRIPTION	UNIT PRICE	EXTENDED PRICE
PLEASE SEND INVOICE TO PURCHASING SERVICES AT ABOVE ADDRESS FOR IMMEDIATE PAYMENT				TOTAL ▶	

VENDOR INSTRUCTIONS:

1. INSTRUCT ALL SHIPPERS TO LIST PURCHASE ORDER NUMBER ON ALL BILLS AND PACKAGES.
2. YOUR INVOICE SHOULD ONLY INCLUDE ITEMS NOTED ON THIS ORDER. PLEASE LIST PURCHASE ORDER NUMBER ON INVOICE.
3. PREPAY ALL TRANSPORTATION CHARGES AND INCLUDE THEM ON YOUR INVOICE IF THEY ARE CHARGEABLE. NO C.O.D.'S.
4. TAX EXEMPT: UTAH STATE UNIVERSITY IS A DIVISION OF STATE GOVERNMENT. UTAH STATE SALES TAX EXEMPTION NUMBER IS 50245.

DEPT. SIGNATURE

TITLE

JB Covington
DIRECTOR OF PURCHASING J.B.Covington

VENDOR COPY

Vendor-**V**, Address-**A**, Date-**D**, Account-**C**, ShipAddress-**S** ,

Partno-**P**, Qty-**Q**, Unit price -**U**, Description-**T**, linetotal -**L**,

.

.

.

.

.

Tax- **X**

Total - **$**

Signature **G**

Date **E**

PURCHASE_ORDER (**Pid**, V, A, D, C, S, P, Q, U, T, L, X, $, G, E)

Anomalies in Database: A Consequence of Bad Design

Redundancies: the same information is stored more than once.

Update Anomalies: The multiple copies may lead to updates which become inconsistent.

Insertion Anomalies: Cannot insert some fact unless some other fact is inserted.

Deletion Anomalies: Deleting one fact may delete another.

PURCHASE_ORDER (**PID**, V, A, D, C, S, P, Q, U, T, L, X, $, G, E)

What are the redundancies and/or anomalies in PURCHASE_ORDER?

Functional Dependencies (FDs)

Given attributes X and Y

Y is said to be functionally dependent on X
if a given value for each attribute in X, uniquely
determines the value of the attributes in Y.

X is called the determinant of the functional
dependency (FD) and the FD is denoted as $X \rightarrow Y$.

STDINF(***Name***, ***Course***, *Phone_No*, *Major*, *Prof*, *Grade*)

Name	*Course*	*Phone_No*	*Major*	*Prof*	*Grade*
Jones	353	237-4539	Comp Sci	Smith	A
Ng	329	427-7390	Chemistry	Turner	B
Jones	328	237-4539	Comp Sci	Clark	B
Martin	456	388-5183	Physics	James	A
Dulles	293	371-6259	Decision Sci	Cook	C
Duke	491	823-7293	Mathematics	Lamb	B
Duke	356	823-7293	Mathematics	Bond	in prog
Jones	492	237-4539	Comp Sci	Cross	in prog
Baxter	379	839-0827	English	Broes	C

The key of **STDINF** is (*Name*, *Course*)

$$\{Name \rightarrow Phone\_No;\ Name \rightarrow Major;$$
$$Name, Course \rightarrow Grade;\ Course \rightarrow Prof\}$$

Vendor-**V**, Address-**A**, Date-**D**, Account-**C**, ShipAddress-**S** ,
For each line
Partno-**P**, Qty-**Q**, Unit price -**U**, Description-**T**, linetotal -**L**,
Total Tax- **X** Total for the PO - **$**
PO approval signature **G** Date of PO approval signature **E**

PURCHASE_ORDER (**Pid**, V, A, D, C, S, P, Q, U, T, L, X, $, G, E)

FDs in Purchase Order from common business rules

Pid →ADCSX$GE,

PidP → QU,

QU → L,

P →T

Problems due to Redundancy

- *Redundancy* creates problems associated with relational schemas:
  - ***redundant storage, insert/delete/update anomalies***
- Integrity constraints, e.g., *functional dependencies*, can be used to identify relational schemas with potential anomalies.
- Main refinement technique: *decomposition*
  - Decompose R(ABCD) with R1(ABC) and R2(BCD) .
- Decomposition should be used with care.
  - Decompose or not to that is the question!

Definition: The decomposition of a relation scheme

$\mathbf{R} = (A_1, A_2, ..., A_n)$

is its replacement by a set of relation schemes

$\{\mathbf{R_1}, \mathbf{R_2}, ..., \mathbf{R_m}\}$, such that

$\mathbf{R_i} \subseteq \mathbf{R}$ for $1 \leq i \leq m$ and

$\mathbf{R_1} \cup \mathbf{R_2} \cup ... \cup \mathbf{R_m} = \mathbf{R}$.

Inference Axioms: Assume that we have a relation scheme R(A_1, A_2, A_3, ..., A_n); R is a relation on the scheme R and W, X, Y, Z are subsets of R.

F1: Reflexivity: X → X.

F2: Augmentation: **X → Y** ⊨ **XZ → Y**, and **XZ → YZ**.

F3: Transitivity: X → Y and Y → Z ⊨ X → Z.

F4: Additivity: X → Y and X → Z ⊨ X → YZ.

F5: Projectivity: X → YZ ⊨ X → Y and X → Z.

F6: Pseudo-trans: X → Y and YZ → W ⊨ XZ → W.

⊨ symbol meaning "implies"

R	*A*	*B*	*C*	*D*	*E*
	a1	b1	c2	d1	e1
	a2	b2	c1	d2	e2
	a3	b1	c2	d1	e3
	a3	b3	c3	d3	e4
	a1	b2	c1	d2	e5
	a4	b4	c4	d4	e6
	a3	b2	c1	d2	e7
	a5	b4	c4	d4	e8

Relation R on the scheme **R**(*A*, *B*, *C*, *D*, *E*)

Illustrates the inference axioms.

FDs: B → CD, B → C , C → D, B → D may hold

If **F** is a set of FD's on a relation scheme **R** then $\mathbf{F}^+$, the ***closure*** of **F**, is the smallest set of FD's such that $\mathbf{F}^+ \supseteq \mathbf{F}$ and no FD can be derived from **F** by using the inference axioms, that are not contained in $\mathbf{F}^+$.

If **R** is not specified, then it is assumed to contain all the attributes that appear in **F**.

Suppose R(A, B, C, D):

the FDs that hold on R are:

$F = \{A \to B, B \to C, C \to D\}$

Then F^+ also contain:
$A \to C$, $A \to D$, $B \to D$

Closure of a set of FDs

Example: Let $\mathbf{F} = \{\mathbf{W} \rightarrow \mathbf{X}, \mathbf{X} \rightarrow \mathbf{Y}, \mathbf{W} \rightarrow \mathbf{XY}\}$ then $\mathbf{F}^+$ includes the set $\{\mathbf{W} \rightarrow \mathbf{W}, \mathbf{X} \rightarrow \mathbf{X}, \mathbf{Y} \rightarrow \mathbf{Y}, \mathbf{W} \rightarrow \mathbf{X}, \mathbf{X} \rightarrow \mathbf{Y}, \mathbf{W} \rightarrow \mathbf{XY}, \mathbf{W} \rightarrow \mathbf{Y}\}$. The first three FD's follow from axiom **F1;** the next three FD's are in **F**, and hence in $\mathbf{F}^+$. Since $\mathbf{W} \rightarrow \mathbf{XY}$ then by axiom **F5** $\mathbf{W} \rightarrow \mathbf{X}$ and $\mathbf{W} \rightarrow \mathbf{Y}$. However, $\mathbf{F}^+$ does not contain a FD, e.g. $\mathbf{W} \rightarrow \mathbf{Z}$, since **Z** is not contained in the set of attributes that appear in **F**.

Functional Dependencies (FDs)

- An FD is a statement about *all* allowable relations on a relational schema.
  - Must be identified based on semantics of application (not from an instance *r* of a relation on the schema R)
  - Given some allowable instance *r* of R, we can check if it violates some FD *f*, but we cannot tell if *f* holds over R!
- K is a candidate key for R means that K → R
- We require the candidate keys to be minimal
- If K’ ⊂ K then K’ ¬ → R
  - However, X → R does not require X to be *minimal(superkey)*!

FDs again!

- Consider relation

  HrEmps (<u>*Sin*</u>, *Name, Grade, Rate*, *HrWrk*):

  let us denote it by listing the attributes simply using: SNGRH

  - The schema is a plan for the co-occurrence of the *set* of attributes {S,N,G,R, H}.
  - We may alternately refer to this set of attributes by using just the relation scheme name. (e.g., HrEmps for {SNGRH})
  - In implementation, we usually have only in relation on each relation scheme
- Some FDs on Hourly_Emps:
  - *Sin* is the candidate key: S → SNGRH
  - *Grade* determines Rate (*hourly wages)*: G → R (*transitive FD*)

Example

❖ Problems due to G → R :

Update anomaly: Can we change R in just the 1st tuple of SNGRH? (e.g., change value of R to 11!)

Insertion anomaly: What if we want to insert an employee and don't know the hourly wage for her grade? Also, we can't insert a rate for a grade unless we have an employee with that grade!

Deletion anomaly: If we delete all employees with grade 35, we lose the information about the wage for this grade!

Emps

S	N	G	R	H
123-223-666	Evan	48	10	40
231-315-368	Lalonde	22	8	30
131-243-650	Letitia	35	9	30
434-263-751	Drew	35	9	32
612-674-134	Ma	48	10	40

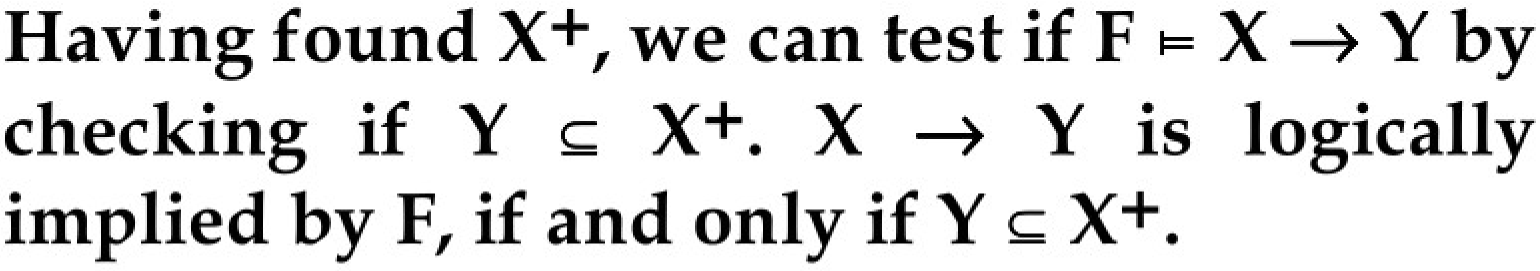
Emp

S	N	G	H
123-223-666	Evan	48	40
231-315-368	Lalonde	22	30
131-243-650	Letitia	35	30
434-263-751	Drew	35	32
612-674-134	Ma	48	40

Wages

G	R
48	10
35	9
22	8

Definition: The closure of X under a set of functional dependencies F and written as X^+, is the set of attributes $\{A_1, A_2, .., A_m\}$ such that the FD $X \rightarrow A_i$ for $A_i \in X^+$ follows from F by the inference axioms for functional dependencies.



Having found X^+, we can test if $F \models X \rightarrow Y$ by checking if $Y \subseteq X^+$. $X \rightarrow Y$ is logically implied by F, if and only if $Y \subseteq X^+$.

Lemma: $\mathbf{F} \vDash \mathbf{X}\rightarrow\mathbf{Y}$ if and only if $\mathbf{Y} \subseteq \mathbf{X}^{+}$.

Proof: Suppose that $\mathbf{Y} \subseteq \mathbf{X}^{+}$. Then by the definition of X^{+}, $\mathbf{X} \rightarrow A$ can be derived from **F** using the inference rules, for each $A \in \mathbf{Y}$.

Now, by the soundness of these rules,

$\mathbf{F} \vDash \mathbf{X} \rightarrow A$ for each $A \in \mathbf{Y}$

and by the additivity rule,

$F \vDash \mathbf{X} \rightarrow \mathbf{Y}$.

Now, suppose that $\mathbf{F} \vDash \mathbf{X} \rightarrow \mathbf{Y}$. Then by completeness of the inference rules, $\mathbf{X} \rightarrow \mathbf{Y}$ can be derived from **F** using them. By projectivity, $X \rightarrow A$ can be derived for each $A \in \mathbf{Y}$.

This clearly implies that $\mathbf{Y} \subseteq \mathbf{X}^{+}$ by the definition of $\mathbf{X}^{+}$.

Algorithm to compute X+

```
$X^+ := X$;    /* initialize $X^+$ to $X$ */
change := true;
while change {
    change := false;
    for each FD $W \to Z$ in F{
        if $W \subseteq X^+$ and $Z \not\subseteq X^+$ then {
                $X^+ := X^+ \cup Z$;
                change := true;}
        }
    }
(* $X^+$ now contains the closure of $X$ under F *)
```

$F=\{A \to B, C \to D, E \to AD, D \to BE\}$

Then $F \models AC \to BDE$

$\because AC+ = AC$ ➲$_{A \to B}$ ABC

➲$_{C \to D}$ ABCD

➲$_{D \to BE}$ ABCDE

Also $AC \to B$,

$AC \to D$,

$AC \to E$

Membership Algorithm

Input: A set of Fds **F**, and the FD $\mathbf{X} \rightarrow \mathbf{Y}$.

Output: Is $\mathbf{X} \rightarrow \mathbf{Y} \in \mathbf{F}^{+}$ or not?

Body:

```
Compute $\mathbf{X}^{+}$
    if $\mathbf{Y} \subseteq \mathbf{X}^{+}$. then $\mathbf{X} \rightarrow \mathbf{Y} \in \mathbf{F}^{+}$ := true;
            else $\mathbf{X} \rightarrow \mathbf{Y} \in \mathbf{F}^{+}$ := false;
```

Definition: Given two sets of FD's **F** and **G** over a relation scheme **R**. **F** and **G** are equivalent (i.e., $\mathbf{F} \equiv \mathbf{G}$) if the closure of **F** is identically equal to the closure of **G** (i.e., $\mathbf{F}^+ = \mathbf{G}^+$). If **F** and **G** are equivalent then **F covers G** and **G covers F**.

Definition: Given a set of FD's **F**, we say that it is **non-redundant** if no proper subset **F'** of **F** is equivalent to **F**, i.e., no **F'** exists such that $\mathbf{F'}^+ = \mathbf{F}^+$.

Title: Algorithm: Non-redundant cover

Input: A set of FD's **F**

Output: A non-redundant cover of **F**

Body

```
G := F; // initialize G to F
for each FD $X \rightarrow Y$ in G do
        if $X \rightarrow Y \in \{F - (X \rightarrow Y)\}^+$
                // i.e. $\{F-(X \rightarrow Y)\} \models X \rightarrow Y$
            then $F := \{F - (X \rightarrow Y)\}$;
G := F; // G is the non-redundant cover of F
end;
```

If **F** = $\{A \rightarrow BC, CD \rightarrow E, E \rightarrow C, D \rightarrow AEH,$

$ABH \rightarrow BD, DH \rightarrow BC\}$

then a non-redundant cover for **F** is:

$$\{A \rightarrow BC,\ \ E \rightarrow C, D \rightarrow AEH,\ \ ABH \rightarrow BD\}.$$

CD+ under (F- $CD \rightarrow E$) is $CDAEHB$ and it includes E *the* RHS.

Hence $CD \rightarrow E$ is redundant.

Definition: A set of functional dependencies $\mathbf{F_c}$ is a ***canonical cover*** if every FD in $\mathbf{F_c}$ satisfies the following :

- *each* FD *in* $\mathbf{F_c}$ *is simple*, (recall that in a simple FD the right hand side has a single attribute i.e., each FD is of the form $\mathbf{X} \rightarrow A$);

- for no FD $\mathbf{X} \rightarrow A$ with $\mathbf{Z} \subset \mathbf{X}$ is $\{(\mathbf{F_c} - (\mathbf{X} \rightarrow A)) \cup (\mathbf{Z} \rightarrow A)\} \rightarrow \mathbf{F_c}$. In other words the left hand side of each FD does not have any extraneous attributes or the *FD's in* $\mathbf{F_c}$ *are left reduced*;

- *no* FD $\mathbf{X} \rightarrow$ A *is redundant* i.e., $\{ \mathbf{F_c} - (\mathbf{X} \rightarrow A)\}$ does not logically imply $\mathbf{F_c}$.

A canonical cover is sometimes called ***minimal***.

Given a set **F** of functional dependencies we can find a canonical set $\mathbf{F_c}$; Obviously $\mathbf{F_c}$ covers **F**.

Example: If **F** = {$A \rightarrow BC$, $CD \rightarrow E$, $E \rightarrow C$, $D \rightarrow AEH$, $ABH \rightarrow BD$, $DH \rightarrow BC$}

then a non-redundant cover for **F** is { $A \rightarrow BC$, $E \rightarrow C$, $D \rightarrow AEH$, $ABH \rightarrow BD$ }:

and the canonical cover is {$A \rightarrow B$, $A \rightarrow C$, $E \rightarrow C$, $D \rightarrow A$, $D \rightarrow E$, $D \rightarrow H$, $AH \rightarrow D$}.

Definition: Given a relation scheme **R** $\{A_1A_2A_3 \ldots A_n\}$,and a set of functional dependencies **F**, a **key** **K** of **R** is a subset of **R** such that the following are satisfied:

- **K** → $A_1A_2A_3 \ldots A_n$ is in $\mathbf{F}^+$
- For any **Y** ⊂ **K**, **Y** → $A_1A_2A_3 \ldots A_n$ is not in $\mathbf{F}^+$

Given R(A, B, C, D) with F{C→ D, C → A, B → C}

Find C^+ the closure of C under F.

C^+ is initialized to C

Using C→ D we augment C^+ to CD

Using C → A, we augment C^+ to CDA

No other change is possible; hence closure of C under F is:

CDA

Find the candidate key for R.

The closure of B under F is BCDA

Hence B is a candidate key.

Given R(A, B, C, D) with F{D → A, B → C}

Find C^+ the closure of C under F.

C^+ is initialized to C

Since C doesn’t appear on the RHS of any FDs in F,

no change is possible; hence closure of C under F is: C

Find the candidate key for R.

The closure of D under F is DA

The closure of B under F is BC

Since neither of the determinants are possible candidate keys, However, BD → ABCD

Hence BD is a candidate key.

Example: If **R** (*ABCDEH*) and **F** = {*A* → *BC*, *CD* → *E*, *E* → *C*, *D* → *AEH*, *ABH* → *BD*, *DH* → *BC*},

Attributes in

Left only:	Left & Right	Right only
none	A, B, C, D, E, H	none

Closure of one attribute

$A^+ = ABC$

$B^+ = B$

$C^+ = C$

$D^+ = DAEHBC$

$E^+ = EC$

$H^+ = H$

Closure of two attributes not involving key already found

$AB^+ = ABC$

$AC^+ = ABC$

$AE^+ = ABEC$

$AH^+ = ABCHDE$

$BC^+ = BC$

$BE^+ = BEC$

$BH^+ = BH$

$CE+ = CE$

$CH^+ = CH$

$EH^+ = ECH$

D is a candidate key of **R** since *D* → *ABCDEH* is in **F+**

No other single attribute candidate key.

Also, *AH* is candidate key since AH+= under **F** is *ABCDEH*.

Since D is only in one RHS, the key must include D or AH

No other keys:

Superkeys are DX or AHX where X ⊆ **R**

Full Functional Dependency

Definition: Given a relational scheme **R** and a FD **X** → **Y**, then **Y** is ***fully functionally dependent*** on **X** if there is no **Z**, where **Z** is a proper subset of **X** such that **Z** → **Y**.

Thus, the dependency **X** → **Y** is left reduced, if there are no extraneous attributes in the left hand side of the dependency.

Example: Given **R** $(ABCDEH)$ and **F** = $\{A \to BC, CD \to E, E \to C, D \to AEH, ABH \to BD, DH \to BC\}$

The FD $ABH \to BD$ is not left reduced since $A \to B$ allows us to eliminate B from the LHS. Also, in $AH \to B$ is not left reduced since we have $A \to B$ and we can thus eliminate it. So the FD $ABH \to BD$ can be replaced by simply $AH \to D$

Example: In the relation scheme **R** (*ABCDEH*) with the FD's,

F = $\{A \rightarrow BC,\ CD \rightarrow E,\ E \rightarrow C,\ CD \rightarrow AH,\ ABH \rightarrow BD,\ DH \rightarrow BC\}$, the dependency $A \rightarrow BC$ is left reduced and *BC* is fully functionally dependent on *A*. However, the functional dependency $ABH \rightarrow D$, is not left reduced, the attribute *B* being extraneous in this dependency.

Definition: An attribute *A* in a relation scheme **R** is a **prime attribute** or simply **prime**, if *A* is part of any candidate key of the relation. If *A* is not a part of any candidate key of **R**, *A* is called a **nonprime attribute** or simply **nonprime**.

Example: If **R** (*ABCDEH*) and **F** = {$A \rightarrow BC$, $CD \rightarrow E$, $E \rightarrow C$, $AH \rightarrow D$}; then *AH* is the only candidate key of R. The attributes *A* and *H* are prime, and the attributes *B*, *C*, *D*, and *E* are nonprime.

Definition: Given a relation scheme **R** with the functional dependencies **F** defined on the attributes of **R**. Let **K** be a candidate key. If **X** is a proper subset of **K**, and if $\mathbf{F} \models \mathbf{X} \rightarrow A$, then, *A* is said to be ***partially dependent*** on **K**.

In the relation scheme

STUDENT_COURSE_INFO(***Name***, ***Course***, *Grade*, *Phone_No*, *Major*, *Course_Dept*) with the FD's,
F = {*Name* → *Phone_NoMajor*, *Course* → *Course_Dept,* *NameCourse* → *Grade*}.

Here *NameCourse* is a candidate key,
Name and *Course* are prime attributes.
Grade is fully functionally dependent on the candidate key.

Phone_No, *Course_Dept*, and *Major* are partially dependent on the candidate key.

Transitive Dependency

Definition: Given a relation scheme **R** with the functional dependencies **F** defined on the attributes of **R**.

Let **X** and **Y** be subsets of **R** and let *A* be an attribute of **R** such that $\mathbf{X} \not\subset \mathbf{Y}$, $A \not\subset \mathbf{XY}$.

If the set of functional dependencies $\{\mathbf{X} \to \mathbf{Y}, \mathbf{Y} \to A\}$ is implied by **F** (i.e., $\mathbf{F} \models \mathbf{X} \to \mathbf{Y} \to A$ and $\mathbf{F} \neg\models \mathbf{Y} \to \mathbf{X}$),

then *A* is ***transitively dependent*** on **X**.

In the relation scheme

Employee(*Emp_Name*, *Department*, *Manager*)

with the function dependencies

F = {*Emp_Name* → *Department*, *Department* → *Manager*},

Emp _Name is the key and *Manager* is transitively

Dependent on the key since

Emp_Name → *Department* → *Manager*.

Content Preserving: the original relation can be derived from the decomposed relations (lossless join decomposition)

Dependency Preserving: the original set of constraints can be derived from the dependencies in the decomposed relations.

Free of interrelation join constraints: if there are no dependencies that can only be derived from the join of two or more decomposed relations

Definition: An *unnormalized relation contains **nonatomic** values.*

Definition: A relation scheme is said to be in the **first normal form** (**1NF**) if the values in the domain of each attribute of the relation are atomic.

1NF has NO NON-ATOMIC ATTRIBUTES.

Definition: A relation scheme **R**<**S**, **F**> is in the **second normal form** (**2NF**) if all non-prime attributes are fully functionally dependent on the relation key(s).

2NF has NO PARTIAL DEPENDENCY

Assignment(Emp#, Name, Dept, Proj#, Hours, Lab)

Emp# → NameDept, Proj# → Lab,

Emp#Proj# → Hours

123	Smith	D1	P1	5	L1
			P2	30	L1
234	Ma	D2	P1	20	L1
			P3	10	L2
			P4	5	L3
345	Russo	D1	P1	35	L1

Example of an unnormalized (non-normal form) relation

123	Smith	D1	P1	5	L1
123	Smith	D1	P2	30	L1
234	Ma	D2	P1	20	L1
234	Ma	D2	P3	10	L2
234	Ma	D2	P4	5	L3
345	Russo	D1	P1	35	L1

Assignment(Emp#, Name, Dept, Proj#, Hours, Lab)
Emp# → NameDept, Proj# → Lab,
Emp#Proj# → Hours

123	Smith	D1
123	Smith	D1
234	Ma	D2
234	Ma	D2
234	Ma	D2
345	Russo	D1

P1	L1
P2	L1
P3	L2
P4	L3

123	P1	5
123	P2	30
234	P1	20
234	P3	10
234	P4	5
345	P1	35

Assignment(Emp#, Name, Dept, Proj#, Hours, Lab)
Emp# → NameDept, Proj# → Lab, Dept → Lab
Emp#Proj# → Hours

123	Smith	D1	P1	35	L1
234	Ma	D2	P2	35	L2
345	Russo	D1	P3	35	L1

One can't say that there is no anomaly from the contents in the database at a given point in time!

Definition: A relation scheme **R**<**S**,**F**> is in the **third normal form**(**3NF**)if for all nontrivial FD in **F+** of the form **X** → *A*, either **X** contains a key (i.e., **X** is a superkey) or *A* is a prime attribute.

A database scheme is in the third normal form if every relation scheme included in the database scheme is in the third normal form.

3NF HAS NO TRANSITIVE DEPENDENCY

Lossless Join Decomposition

Definition: A decomposition of a relation scheme **R <S,F>** into the relation schemes $\mathbf{R_i}$ ($1 \le i \le n$) is said to be ***lossless join decomposition*** or simply ***lossless*** if for every relation r(**R**) that satisfies the FD's in **F**, the natural join of the projections of r gives the original relation R: i.e.,

$$r = \Pi_{\mathbf{R1}}(r) \bowtie \Pi_{\mathbf{R2}}(r) \bowtie \ldots. \bowtie \Pi_{\mathbf{Rn}}(r)$$

If $r \subset \Pi_{\mathbf{R1}}(r) \bowtie \Pi_{\mathbf{R2}}(r) \bowtie \ldots. \bowtie \Pi_{\mathbf{Rn}}(r)$ then the decomposition is called ***lossy***.

$r \subseteq \Pi_{\mathbf{R1}}(r) \bowtie \Pi_{\mathbf{R2}}(r) \bowtie \ldots. \bowtie \Pi_{\mathbf{Rn}}(r)$ is always true.

Assignment(Emp#, Name, Dept, Proj#, Hours, Lab)
Emp# → NameDept, Proj# → Lab,
Emp#Proj# → Hours

123	Smith	D1
123	Smith	D1
234	Ma	D2
234	Ma	D2
234	Ma	D2
345	Russo	D1

P1	L1
P2	L1
P3	L2
P4	L3

123	P1	5
123	P2	30
234	P1	20
234	P3	10
234	P4	5
345	P1	35

Join the first two relations:
Creates extraneous tuples

When we join the third relation, the extraneous tuples are eliminated! Hence, the decomposition is lossless

123	Smith	D1	P1	L1
123	Smith	D1	P2	L1
123	Smith	D1	P3	L2
124	Smith	D1	P4	L3
234	Ma	D2	P1	L1
234	Ma	D2	P2	L1
234	Ma	D2	P3	L2
234	Ma	D2	P4	L3
345	Russo	D1	P1	L1
345	Russo	D1	P2	L1
345	Russo	D1	P3	L2
345	Russo	D1	P4	L3

123	P1	5
123	P2	30
234	P1	20
234	P3	10
234	P4	5
345	P1	35

Assignment(Emp#, Name, Dept, Proj#, Hours, Lab)

Emp# → NameDept, Proj# → Lab, Emp#Proj# → Hours

123	Smith	D1	P1	5	L1
123	Smith	D1	P2	30	L1
234	Ma	D2	P1	20	L1
234	Ma	D2	P3	10	L2
234	Ma	D2	P4	5	L3
345	Russo	D1	P1	35	L1

123	Smith	D1	L1
234	Ma	D2	L1
234	Ma	D2	L2
234	Ma	D2	L3
345	Russo	D1	L1

P1	5	L1
P2	30	L1
P1	20	L1
P3	10	L2
P4	5	L3
P1	35	L1

A lossy join decomposition

123	Smith	D1	P1	5	L1
123	Smith	D1	P1	20	L1
123	Smith	D1	P1	35	L1
123	Smith	D1	P2	30	L1
234	Ma	D2	P1	5	L1
234	Ma	D2	P1	20	L1
234	Ma	D2	P1	35	L1
234	Ma	D2	P2	30	L1
234	Ma	D2	P3	10	L2
234	Ma	D2	P4	5	L3
345	Russo	D1	P1	5	L1
345	Russo	D1	P1	20	L1
345	Russo	D1	P1	35	L1
345	Russo	D1	P2	30	L1

Definition: Given a relation scheme **R**<**S**,**F**> where **F** is the associated set of functional dependencies on the attributes in **S**. Consider that **R** is decomposed into the relation schemes $\mathbf{R_1}$, $\mathbf{R_2}$, ... , $\mathbf{R_n}$ with the functional dependencies $\mathbf{F_1}$, $\mathbf{F_2}$,.. , $\mathbf{F_n}$.

Then this decomposition of **R** is ***dependencies preserving***, if the closure of **F'** (where $\mathbf{F'} = \mathbf{F_1} \cup \mathbf{F_2} \cup \ldots \cup \mathbf{F_n}$) is identical to $\mathbf{F^+}$ (i.e., $\mathbf{F'^+} \equiv \mathbf{F^+}$).

Theorem : A decomposition of relation scheme **R** <(X, Y, Z), F> into say $\mathbf{R}_1$<(**X**, **Y**), $\mathbf{F}_1$> and $\mathbf{R}_2$<(**X**,**Z**), $\mathbf{F}_2$> is:

(i) dependency preserving if every functional dependency in **R** can logically derived from the functional dependencies of $\mathbf{R}_1$ and $\mathbf{R}_2$ i.e., $(\mathbf{F}_1 \cup \mathbf{F}_2)^+ = \mathbf{F}^+$, and

(ii) is lossless if the common attributes **X** of $\mathbf{R}_1$ and $\mathbf{R}_2$ form a superkey of at least one of these i.e., $\mathbf{X} \rightarrow \mathbf{Y}$ or $\mathbf{X} \rightarrow \mathbf{Z}$.

```
for each decomposed relation $\mathbf{R}_i$ do
  if an attribute $A_j$ is included in $\mathbf{R}_i$,
   then TABLE_LOSSY(i,j) := $\alpha_{Aj}$
   else TABLE_LOSSY(i,j) := ß$_{iAj}$
change := true
while (change) do
  for each FD $\mathbf{X} \rightarrow \mathbf{Y}$ in F do
     if rows i and j exist such that the $\alpha_r$ symbol appears in
             each column corresponding to the attributes of $\mathbf{X}$
             then if one of the symbol in the $\mathbf{Y}$ column is $\alpha_r$ , the other ß$_r$
              then make replace ß$_r$ with $\alpha_r$
               else if the symbols are ß$_{pm}$ and ß$_{qm}$
                                          then make both of them, say, ß$_{pm}$;
        else change := false
i := 1
If there is a row will all $\alpha$ then the decomposition is lossless
```

Example: $\mathbf{R}(A,B,C,D)$ with the functional dependencies $\mathbf{F}$ $\{A \rightarrow B, A \rightarrow C, C \rightarrow D\}$. Consider the dependence preserving decomposition of **R** into $\mathbf{R}_1(A,B,C)$ and $\mathbf{R}_2(C,D)$.

	A	B	C	D	A	B	C	D
$\mathbf{R}_1$	α_A	α_B	α_C	β_{1D}	α_A	α_B	α_C	α_D
$\mathbf{R}_2$	β_{2A}	β_{2B}	α_C	α_D	β_{2A}	β_{2B}	α_C	α_D

Example: R(A, B, C, D, E) with the functional dependencies F $\{AB \rightarrow CD, A \rightarrow E, C \rightarrow D\}$. Then the decomposition of **R** into $\mathbf{R}_1(A,B,C)$ and $\mathbf{R}_2(B,C,D)$ and $\mathbf{R}_3(C,D,E)$ is lossy.

$\mathrm{F}\ \{AB \rightarrow CD, A \rightarrow E, C \rightarrow D\}$.

	A	B	C	D	E
$R_1(A,B,C)$	α	α	α	β	β
$R_2(B,C,D)$	β	α	α	α	β
$R_3(C,D,E)$	β	β	α	α	α

$C \rightarrow D$

	A	B	C	D	E
$R_1(A,B,C)$	α	α	α	~~β~~α	β
$R_2(B,C,D)$	β	α	α	α	β
$R_3(C,D,E)$	β	β	α	α	α

No further changes – no row with all α

Hence , lossy decomposition!

Algorithm to check if a decomposition is dependency preserving

Input: A relation scheme and a set F of FDs: a projection (R_1, R_2, ..., R_n) of **R** with the FDs (F_1, F_2, ..., F_n).

Output: Whether the decomposition is dependency preserving or not.

Body:

```
$F'^+\_=\_F^+$ := *true*;
$F'$ := ø;
*for* i:= 1 **to** n *do*
   $F' := F' \cup F_i$;
*for each* FD $X \rightarrow Y \in F$  *and while* ($F'^+\_=\_F^+$) *do*
          // compute $X'^+$, the closure of X under F'
   *if* $Y \not\subset X'^+$ *then* $F'^+\_=\_F^+$ := *false*;
```

Example : Consider the relation scheme **R**(A,B,C,D) with the FDs **F** = {$A \rightarrow B$, $A \rightarrow C$, $C \rightarrow D$}. Here, the decomposition of **R** into $\mathbf{R_1}$<(A,B,C), {$A \rightarrow B$, $A \rightarrow C$} and $\mathbf{R_2}$<(C,D), {$C \rightarrow D$}> is dependence preserving, since in this case each FD in **F** is included in **F'** (where **F'** = $\mathbf{F_1} \cup \mathbf{F_2}$).

Example: Consider the relation Student_Advisor(*Name*, *Dept*, *Advisor*) with the FDs **F** = {$N \rightarrow D$, $N \rightarrow A$, $A \rightarrow D$}. Here, its decomposition into S_Pr<(N, A), {$N \rightarrow A$}>, and D_A <(D, A), {$A \rightarrow D$}> is dependence preserving, since $N \rightarrow D$ is implied by ($N \rightarrow A$) $\cup$ ($A \rightarrow D$); in addition the decomposition is lossless.

Example: Consider $\mathbf{R}(A,B,C,D)$ with the FDs

$\mathbf{F}\ \{A \rightarrow B,\ A \rightarrow C,\ C \rightarrow D\}$ and its decomposition into

$\mathbf{R}_1(A,B,C)$ with the FDs $\mathbf{F}_1 = \{A \rightarrow B,\ A \rightarrow C\}$ and

$\mathbf{R}_2(C,D)$ with the FDs $\mathbf{F}_2 = \{C \rightarrow \mathrm{D}\}$.

This decomposition is dependence preserving since all the original FD's can be logically derived from $\mathbf{F}_1$ and $\mathbf{F}_2$.

Example: $\mathbf{R}(A,B,C,D)$ with the FDs F $\{A \rightarrow B,\ A \rightarrow C,\ A \rightarrow D\}$
is decomposed into $\mathbf{R}_1(A,B,D)$ with the FDs$\mathbf{F}_1 = \{A \rightarrow B,\ A \rightarrow D\}$
and $\mathbf{R}_2(B,C)$ with the FDs $\mathbf{F}_2 = \{\ \}$
is not dependence preserving since the FD
$A \rightarrow C$ is not implied by any FD's in $\mathbf{R}_1$ or $\mathbf{R}_2$.

Example The decomposition of the relation Concentration (Student, Major_or_Minor, Dept, Advisor), with the FDs $\{SM_mF_s \rightarrow A, A \rightarrow F_s\}$ into the relations SM_mA and F_sA is not FD preserving since **F'** = $A \rightarrow F_s$ and the FD $SM_mF_s \rightarrow A$ is not implied by **F'**.

Third Normal Form Decomposition Algorithm

Input: A relation Scheme **R**, a set of Canonical FDs $\mathbf{F_c}$, and **K** a candidate key of **R**.(K must have any attributes $\not\in$ $\mathbf{F_c}$)

Output: A collection of third normal form relation schemes ($\mathbf{R_1}$, $\mathbf{R_2}$, … $\mathbf{R_i}$) which are dependency preserving and lossless.

```
i := 0
*if* there is a dependency $\mathbf{X} \rightarrow \mathbf{Y}$ in $\mathbf{F_c}$ such that all the
attributes that remain in R are included in it{
        i:= i+1;  output R as $\mathbf{R_i}$\{ $\mathbf{X}$, $\mathbf{Y}$\};}
  *else{ for each* FD $\mathbf{X} \rightarrow A$ *in* $\mathbf{F_c}${
              i:= i+1; form $\mathbf{R_i}$<{$\mathbf{X}$,$A$ }, $\mathbf{F_i}${ $\mathbf{X} \rightarrow A$ }>}
Replace (<($\mathbf{X}$,$A$), {$\mathbf{X} \rightarrow A$}> and <($\mathbf{X}$,$B$), {$\mathbf{X} \rightarrow B$}> with
        <($\mathbf{X}$,$AB$), {$\mathbf{X} \rightarrow AB$}>)
              *if* $\mathbf{F_j}$ for $1 \leq j \leq i$  not satisfies $\mathbf{K} \subseteq \mathbf{X}$ {
                       i := i+1; form $\mathbf{R_i}$ { K }
```

SHIPPING(*Ship*, *Capacity*, *Date*, *Cargo*, *Value*)

Ship → *Capacity*,
ShipDate → *Cargo*,
CargoCapacity → *Value*

The given set of FD's is in canonical form.

A candidate key of the relation is *ShipDate*.

Decompose into:

$\mathbf{R}_1$(*Ship*, *Capacity*) with the FD: *Ship* → *Capacity*,
$\mathbf{R}_2$(*Ship*, *Date*, *Cargo*) with the FD: *ShipDate* → *Cargo*,
$\mathbf{R}_3$(*Cargo*, *Capacity*, *Value*) with the FD: *CargoCapacity* →*Value*

Consider the relation scheme **Student_info**($Student(S)$, $Major(M)$, $Student\_Department(S_d)$, $Advisor(A)$, $Course(C)$, $Course\_Department(C_d)$, $Grade(G)$, $Professor(P)$, $Prof\_Department(P_d)$, $Room(R)$, $Day(D)$, $Time(T)$) with

$\{S \rightarrow M, S \rightarrow A, M \rightarrow S_d, S \rightarrow S_d, A \rightarrow S_d, C \rightarrow C_d, C \rightarrow P,$
$P \rightarrow P_d, RTD \rightarrow C, RTD \rightarrow P, TPD \rightarrow R, TSD \rightarrow R,$
$TDC \rightarrow R, TPD \rightarrow C, TSD \rightarrow C, SC \rightarrow G\}$

Redundant FDs $\{S \rightarrow Sd, RTD \rightarrow P, TDC \rightarrow R, TPD \rightarrow C, TSD \rightarrow R\}$.

The primary key is TSD.

3NF decomposition is: <$\mathbf{R_1}(SMA)$, $\{S \rightarrow MA\}$>; <$\mathbf{R_2}(MSd)$,
$\{M \rightarrow Sd\}$>, < $\mathbf{R_3}(ASd)$;, $\{A \rightarrow Sd\}$>; < $\mathbf{R_4}(CCdP)$, $\{C \rightarrow CdP\}$>; < $\mathbf{R_5}(PPd)$,
$\{P \rightarrow Pd\}$>; < $\mathbf{R_6}(RTDC)$, $\{RTD \rightarrow C\}$>; < $\mathbf{R_7}(TPDR)$ $\{TPD \rightarrow R\}$>; < $\mathbf{R_8}(TSDR)$,
$\{TSD \rightarrow R\}$>; < $\mathbf{R_9}(SCG)$ $\{SC \rightarrow G\}$>.

Name	*Student#*	*Course*	*Grade*
Jones	23714539	353	A
Ng	42717390	329	A
Jones	23714539	328	in prog
Martin	38815183	456	C
Dulles	37116259	293	B
Duke	82317293	491	C
Duke	82317293	353	in prog
Jones	23714539	491	C
Evan	11011978	353	A+
Baxter	83910827	379	in prog

The Grade Relation

Suppose the FDs are ***Student#* → *Name*** and ***Name* → *Student#***?

Is it in 3NF? Any redundancies?

Definition: A normalized relation scheme **R**<**S**,**F**> is in the **Boyce Codd normal form** if for every nontrivial FD in **F+** of the form **X** $\rightarrow A$ where **X** $\subseteq$ **S** and $A \in$ **S**, **X** is a superkey of **R**.

A database scheme is in the BCNF if every relation scheme in the database scheme is in the BCNF.

The relation GRADE is not in the BCNF because of the dependencies *Student#* → *Name* and *Name* → *Student#* are nontrivial and their determinants are not superkeys of GRADE.

Algorithm: Lossless BCNF Decomposition

```
i := 0;
S := { R(U) };
all_BCNF := false;
Find a non-redundant cover F' from F
while ( ¬all_BCNF ){
   if $\exists((X \rightarrow Y) \in F'^{+} \wedge Y \not\subseteq X) \wedge (XY \subseteq R_j) \wedge X \neg\rightarrow R_j${
            i := i+1;
            $<R_i\{X, Y\}, X \rightarrow Y> \cup S$
             $R_j := R_j - Y$;
             }
             else  all_BCNF := true;
}
```

Example: Let us find a BCNF decomposition of the relation:

SHIPPING(*Ship*, *Capacity*, *Date*, *Cargo*, *Value*)

{$S \rightarrow Cap$, $SD \rightarrow Cargo$, $CargoCap \rightarrow V$

There are no redundant FD'S in the set

Since $S \rightarrow Cap$ and since *Ship* $\neg\rightarrow$ **SHIPPING** replace **SHIPPING** with: $\mathbf{R}_1$(*S*, *Cap*) and $\mathbf{R}_2$(*S*, *D*, *Cargo*, *V*).

The decomposition is lossless but not FD preserving: the FD $CargoCap \rightarrow V$ is not implied by {$Ship \rightarrow Cap$, $SD \rightarrow Cargo$}.

A BCNF decomposition which is lossless and FD preserving:

$\mathbf{R}_1$(*Cargo*, *Capacity*, *Value*) with the FD $CargoCapacity \rightarrow Value$,

$\mathbf{R}_2$(*Ship*, *Capacity*) with the FD $Ship \rightarrow Capacity$

$\mathbf{R}_3$(*Ship*, *Date*, *Cargo*) with the FD $ShipDate \rightarrow Cargo$

Given F_1={PersonName → City, Street;
PersonName,CompName → Salary;
CompName → CompCity;
PersonName → MgrName}
and
Given F_2={CompName → CompCity;
PersonName, CompName, CompCity → Salary;
PersonName → City;
PersonName → Street;
PersonName, City → MgrName}

Does F_1 cover F_2?

Given F_1={PersonName → City, Street;
PersonName,CompName → Salary;
CompName → CompCity;
PersonName → MgrName}

Candidate key: PersonNameCompanyName
No redundant attributes on the LHS.
No redundant FDs

$R(P_n, C_n, M_n, C, S, C_c, \$)$ can be decomposed, using the 3NF algorithm (FD preserving and losslessly) into:

$R_1(P_nCS)$, $R_2(P_nC_n\$)$, $R_3(C_nC_c)$, $R_4(P_nM_n)$

Given F_1={PersonName → City, Street;
PersonName,CompName → Salary;
CompName → CompCity;
PersonName → MgrName}

Candidate key: PersonNameCompanyName P_nC_n

No redundant attributes on the LHS.

No redundant FDs

$R(P_n, C_n, M_n, C, S, C_c, \$)$ can be decomposed, using the BCNF algorithm as follows:

$P_nC_nM_nCSC_c\$$

$R_1(P_nCS)$ $P_nC_nM_nC_c\$$

$R_2(P_nC_n\$)$ $P_nC_nM_nC_c$

$R_3(P_nM_n)$ $P_nC_nC_c$

$R_4(C_nC_c)$ P_nC_n

Do not use the FD $P_nC_n \rightarrow \$$
WHY???

P_nC_n is already in LHS of R_2, we can combine it with it(drop it).

Given F_1={PersonName → City, Street;

PersonName,CompName → Salary;

CompName → CompCity;

PersonName → MgrName}

Candidate key: PersonNameCompanyName

No redundant attributes on the LHS.

No redundant FDs

Since P_nC_n is the key we need not use it in decompoing it in the second step*(as shown on the previous slide!!)*

Hence, $R(P_n, C_n, M_n, C, S, C_c, \$)$ can be decomposed, alternatively, using the BCNF algorithm as follows:

$P_nC_nM_nCSC_c\$$

$R_1(P_nCS)$ $P_nC_nM_nC_c\$$

$R_2(P_nM_n)$ $P_nC_nC_c\$$

$R_3(C_nC_c)$ $R_4(P_nC_n\$)$

Given R = <{ABCDEGIJK},
{AB → CDE, E → G, B → G, BG → AIJ, IJ → K}

F = {AB → C	BG → A
AB → D	BG → I
AB → E	BG → J
E → G	IJ → K
B → G}	

F_c = {B → A, B → C, B → D, B → E,
B → I, B → J, IJ → K, E → G}

B is a candidate key.

(ABCDEGIJK)

$F_c = \{B \to A,\ B \to C, B \to D, B \to E,$

$B \to I, B \to J, E \to G, IJ \to K\}$

B is a candidate key.

3NF: Since there is no single FD which includes all attributes in R, we create a relation for each FD:

R1(AB), R2(BC), R3(BD), R4(BE), R5(BI), R6(BJ), R7(EG), R8(IJK)

Combine the relations with the same LHS:

R1’(ABCDEIJ), R7(EG), R8(IJK)

Why did we not include G in R1’ ?

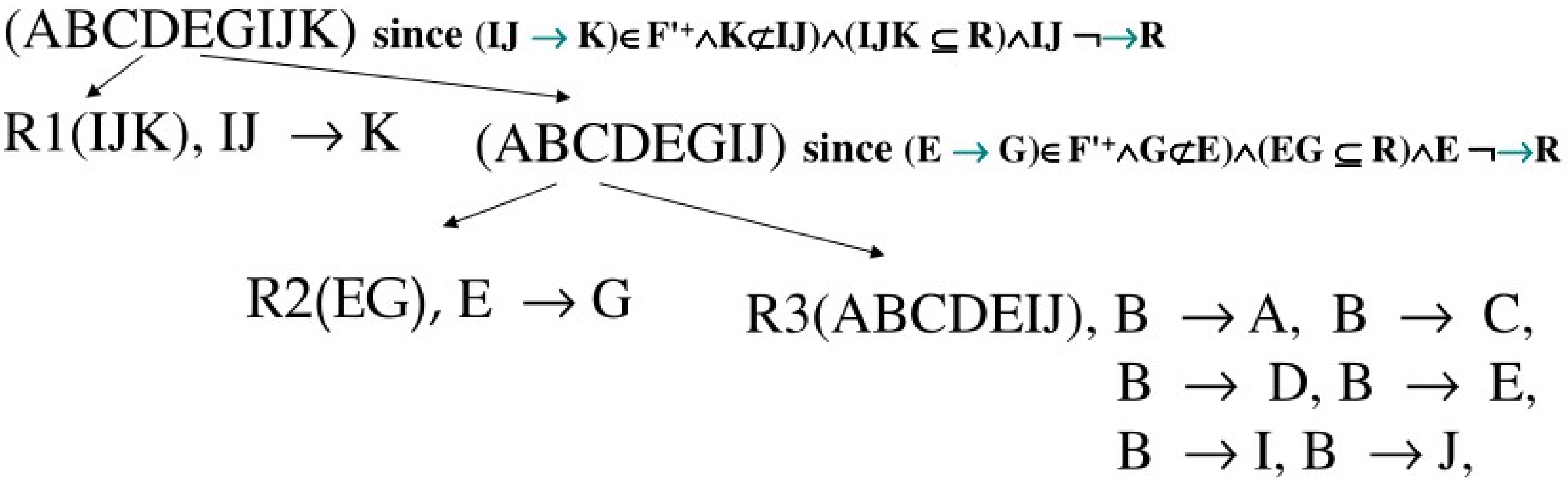
(ABCDEGIJK)

$F_c = \{B \rightarrow A,\ B \rightarrow C, B \rightarrow D, B \rightarrow E,$

$B \rightarrow I, B \rightarrow J, E \rightarrow G, IJ \rightarrow K\}$

B is a candidate key.

BCNF decomposition



(ABCDEGIJK) **since** $(IJ \rightarrow K) \in F'^{+} \wedge K \not\subset IJ) \wedge (IJK \subseteq R) \wedge IJ \neg\rightarrow R$

R1(IJK), $IJ \rightarrow K$

(ABCDEGIJ) **since** $(E \rightarrow G) \in F'^{+} \wedge G \not\subset E) \wedge (EG \subseteq R) \wedge E \neg\rightarrow R$

R2(EG), $E \rightarrow G$

R3(ABCDEIJ), $B \rightarrow A,\ B \rightarrow C,$
$B \rightarrow D, B \rightarrow E,$
$B \rightarrow I, B \rightarrow J,$

Decompose :

Projects<{Employee, Project, Dept,Part, QtyUsed,
HrsWorked},
{Employee,Project → HrsWorked; Project → Dept;
Project,Part → QtyUsed }>

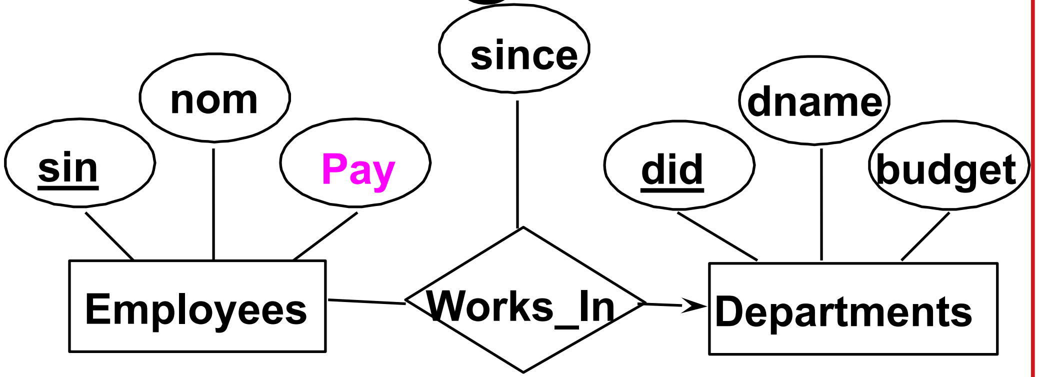
Refining an ER Diagram

1st diagram translated:

Emp(S,N,D,P,S)

Dept(D,Dn,B)

Pay is associated with Emp.



Before:

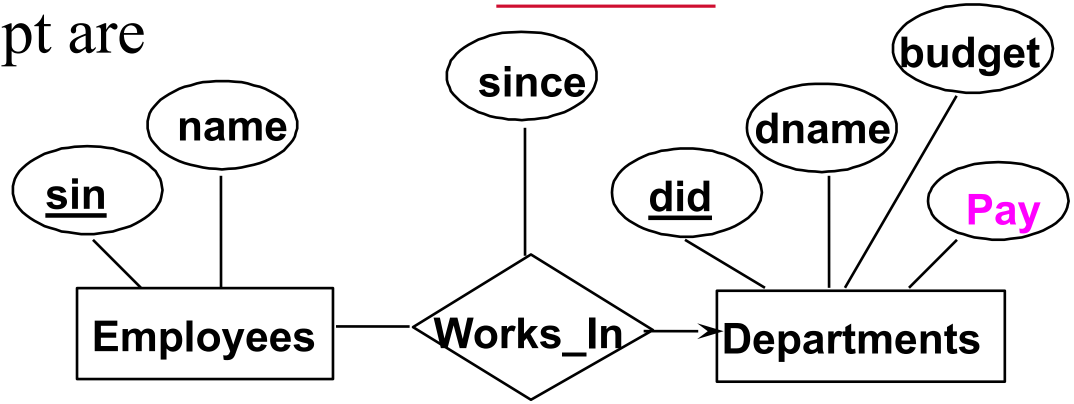
Suppose all workers in a dept are assigned the same pay:

D → P

Redundancy; fixed by:

Emp2(S,N,D,S)

Dept2(D,Dn,B,P)



After:

Normal Forms: Conclusions

-Returning to the issue of schema refinement, the first question to ask is whether any refinement is needed!

-If a relation is in a certain *normal form* (BCNF, 3NF etc.), it is known that certain kinds of problems are avoided/minimized. This can be used to help us decide whether decomposing the relation will help.

- Role of FDs in detecting redundancy:

- Consider a relation R with 3 attributes, ABC.
  - No FDs hold: There is no redundancy here.
  - However if A → B: Then, several tuples could have the same A value, and if so, they'll all have the same B value! We need refinement!

Boyce-Codd Normal Form (BCNF)

- Reln R with FDs *F* is in BCNF if, for all X → A in
  - A ∈ X (called a *trivial* FD), or
  - X contains a key for R.
- In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.
  - No dependency in R that can be predicted using FDs alone.
  - If we are shown two tuples that agree upon the X value,
  - we cannot infer the A value in one tuple from the A
  - value in the other.
  - If example relation is in BCNF, the 2 tuples must be identical (since X is a key).

X	Y	A
x	y1	a
x	y2	?

Third Normal Form (3NF)

- Relation R with FDs *F* is in 3NF if, for all $X \rightarrow A$ in
  - $A \in X$ (called a *trivial* FD), or
  - X contains a key for R, or
  - A is part of some key for R.
- *Minimality* of a key is crucial in third condition above!
- If R is in BCNF, obviously it is also in 3NF.
- If R is in 3NF, some redundancy is possible. It is a compromise, used when BCNF not achievable (e.g., no ``good” decomposition, or performance considerations).
  - *Lossless-join, dependency-preserving decomposition of R into a collection of 3NF relations always possible.*

When is R not in 3NF?

- If 3NF violated by X → A, one of the following holds:
  - X is a subset of some key K(Partial Dep)
    - We store (X, A) pairs redundantly.
  - X is not a proper subset of any key.(Trans. Dep)
    - There is a chain of FDs K → X → A, which means that we cannot associate an X value with a K value unless we also associate an A value with an X value.
- But: even if relation is in 3NF, these problems could arise.
  - (Member,Chalet, Date, Card), M → C, C → M is in 3NF, but for each reservation of member, same (M, C) pair is stored.
- Thus, 3NF is indeed a compromise relative to BCNF.

Decomposition of a Relation Scheme

- Suppose that relation R contains attributes *A1 ... An*. A *decomposition* of R consists of replacing R by two or more relations such that:
  - Each new relation scheme contains a subset of the attributes of R (and no attributes that do not appear in R), and
  - Every attribute of R appears as an attribute of one of the new relations.
- Intuitively, decomposing R means we will store instances of the relation schemes produced by the decomposition, instead of instances of R.

Example Decomposition

- Decompositions should be used only when needed.
  - SNPGRH has FDs S → SNPGRH and G → R
  - Second FD causes violation of 3NF; R values repeatedly associated with G values. Easiest way to fix this is to create a relation GR to store these associations, and to remove R from the main schema:
    - i.e., we decompose SNPGRH into SNPGH and GR
- The information to be stored consists of SNPGRH tuples. If we just store the projections of these tuples onto SNPGH and GR, are there any potential problems that we should be aware of?

Given R(A, B, C, D) with F{C→ D, C → A, B → C}

If R is not in BCNF, decompose it into a set of BCNF relations that preserve the FDs.

B is the candidate key.

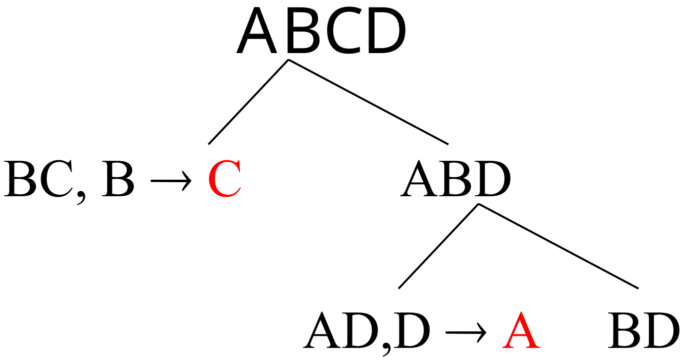
Both C → D and C → A cause BCNF violations.

One way to obtain a (lossless) join preserving decomposition is to decompose R into

AC, BC, and CD.

Given R(A, B, C, D) with F{D → A, B → C}

Here BD is a candidate key. R is in 1NF but not 2NF due to the partial dependencies.
B → C and D → A



ABCD

BC, B → C

ABD

AD,D → A

BD

The decomposition: AD, BC, BD

- obtained by first decomposing R into AD, BCD;
- followed by decomposing BCD into BC and BD

is BCNF and lossless and join-preserving.

Review of Relational Design

©Bipin C. DESAI

Example relation:
EMPLOYEE
(EID, Project, Component,EName,,Building, Room, TelNo)

Note: Keys are underlined.
What are the FDs?
What is the normal form of the relation?

-Only one phone in each room.
R(I, P, C, N, B, R, T)
FD: {I → NTBR, BR → T, T → BR}
Key: IPC
All partial dependencies.
R1(BRT), R2(INT), R3(IPC)

Example relation:
EMPLOYEE
(EID,Project#,Component,Qty,EName,Building,Room,TelNo,Hours)

Note: Keys are underlined.
What are the FDs? What is the normal form of the relation?

-Only one phone in each room.
-There is a m-to-n relationship between projects and components
-Each employee works a number of hours on a project
R(I, P, C, Q, N, B, R, T, H)

Key: IPC
FD: {I → NTBR, BR → T, T → BR, IP → H, PC → Q}

R(I, P, C, Q, N, B, R, T, H)

Key: IPC

FD: F{I → NTBR, BR → T, T → BR, IP → H, PC → Q}

The corresponding F_c is

F_c{I → N, I → T, BR → T, T → BR, IP → H, PC → Q}

A 3NF decomposition is:

R(INT), R(BRT), R(IPH), R(PCQ), R(IPC),

It is also in BCNF!

R(I, P, C, N, B, R, T)

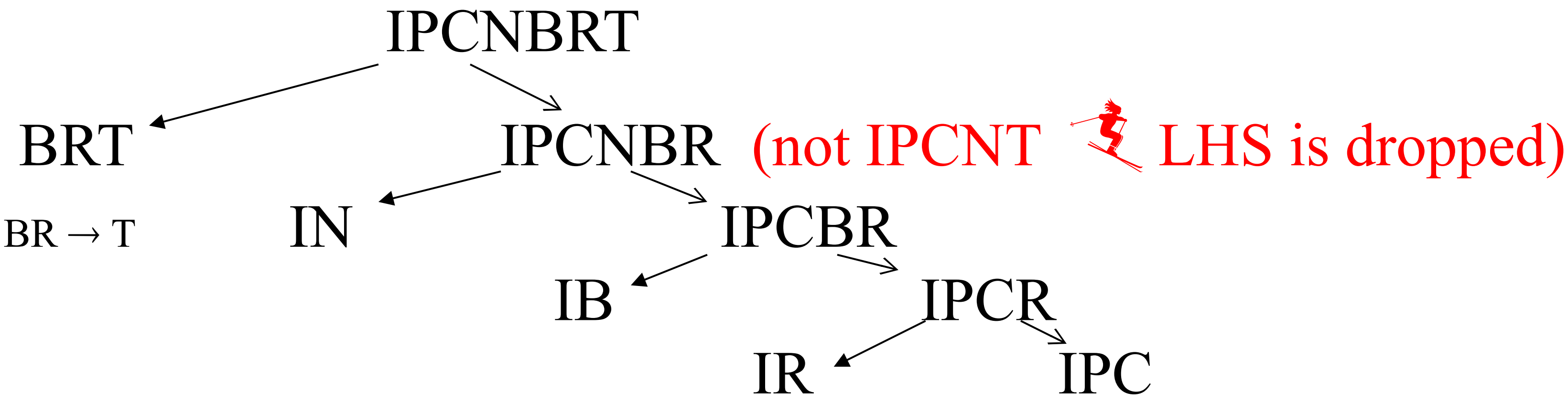
Key: IPC

FD: {I → N, I → T , I → B , I → R, BR → T, T → BR}

3NF decomposition: R1(IN), R2(IT), R3(IB), R4(IR), R5(BRT),
R6(IPC)

Can we combine R1 …. R4?

BCNF decomposition:


IPCNBRT

BRT

IPCNBR (not IPCNT LHS is dropped)

BR → T

IN

IPCBR

IB

IPCR

IR

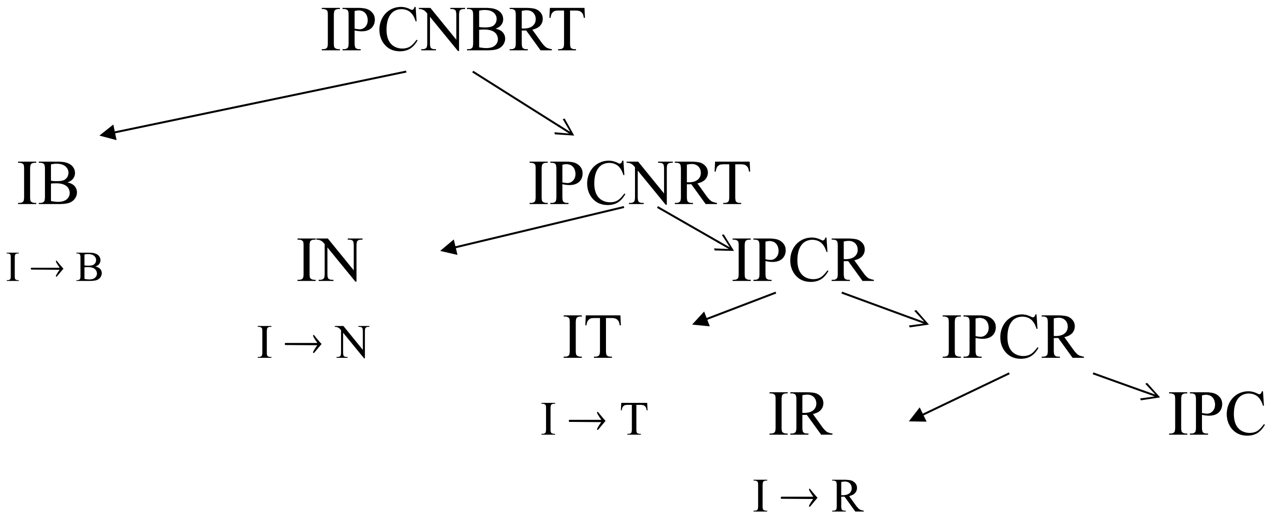
IPC

R(I, P, C, N, B, R, T)

Key: IPC

FD: $\{I \rightarrow N, I \rightarrow T, I \rightarrow B, I \rightarrow R, BR \rightarrow T, T \rightarrow BR\}$

Another BCNF Decomposition which is lossless but NOT FD preserving:



IPCNBRT

IB

I → B

IPCNRT

IN

I → N

IPCR

IT

I → T

IPCR

IR

I → R

IPC

Example: 1NF but not 2NF

ORDER(SuplNo, Address, Distance, PartNo, Price)

Assume each supplier is located in only one Address.

What are the FDs?

What are the anomalies?

PO (SuplNo,, PartNo, Price)

Supplier (SuplNo, Address, Distance)

What are the FDs in each and the normal form of each?

Any anomalies?

Decomposition (into 3NF):

SUPPLIER_Address (SuplNo, Address)

Address_DISTANCE (Address, Distance)

Example (3NF but not BCNF):

Can_Supply (SuplNo, SuplName, Address, PartNo, Price)

Functional Dependencies:

We assume that SuplNo, Address, PartNo are always unique
Thus we have two candidate keys:

(SuplNo, PartNo) and (SuplName, Address, PartNo)

and we have the following dependencies:

(SuplNo, PartNo) → Price
(SuplNo, PartNo) → SuplName, Address
(SuplName, Address, PartNo) → Price
(SuplName, Address, PartNo) → SuplNo
SuplName, Address → SuplNo
SuplNo → SuplName, Address

Decomposition (into BCNF)

SUPPLIER (SuplNo, SuplName)

SUPPLIER_PARTS (SuplNo, PartNo, Quantity)

A relation is in BCNF iff every determinant is a candidate key

BCNF addresses the situations which 3NF does not handle.
In many real DB design the relations in 3NF are also in BCNF.

When is a relation in 3NF not in BCNF:
it has multiple composite candidate keys, and
these candidate keys are non-disjoint
(at least one common attribute)
Example:

Can_Supply (SuplNo, SuplName, Address, PartNo, Price)

Can_Supply is an example of a relation in 3NF but not in BCNF

Can_Supply exhibits the above properties).

The following relation is in 3NF, and also in BCNF:

SUPPLIERS (SuplNo, Suplname, Address, PostalCode)

We assume that each supplier has a unique Suplname, so that SuplNo and Suplname are both candidate keys.

These candidate keys are *not* composite keys and hence the 3NF is also BCNF(all FDs the LHS is a candidate key)

Functional Dependencies:

SuplNo → Address

SuplNo → PostalCode

SuplNo → SuplName

SuplName → SuplNo

SuplName → Address

SuplName → PostalCode

Anomalies *even in a* BCNF relations:

SUPPLIERS (SuplNo, SuplName, Address, PostalCode)

INSERT: We cannot record the Address for a SuplNo without also knowing the SuplName

DELETE: If we delete the row for a given SuplName, we lose the information that the SuplNo is associated with a given Address.

UPDATE: Since SuplName is a candidate key (unique), there are none.

Decomposition:

SUPPLIER_INFO (SuplNo, Address, PostalCode)
SUPPLIER_NAME (SuplNo, SuplName)

$R(ABC)\ F=\{ AB \rightarrow C, C \rightarrow B)$

This is in 3NF but not in BCNF.

There is no need (no way) to decompose this relation!

$R(X,Y, Z)$ with $F=\{XY \rightarrow Z, YZ \rightarrow X, XZ \rightarrow Y\}$

The candidate keys are: XY, YZ and XZ.
This relation is in BCNF since the determinant of each FD is a candidate key!
There is no need (no way) to decompose this relation!

Course Notes

Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

Relational Calculus

Bipin C. Desai

Propositional Logic …

A **proposition** is a statement that is either **true** or **false** (but not both).

In propositional logic, we assume a collection of atomic propositions are given, e.g. $p, q, r, s, t, \ldots.$

p = "COMP5531 *is about databases*"

q = "COMP5531 *is an important course*"

r = "databases *is an important course*"

$\neg p$ = " COMP5531 is not about databases"

$p \wedge q$ = "COMP5531 is about databases and COMP5531 is an important course"

$p \wedge q \rightarrow r$ = "COMP5531 is about databases and COMP5531 is an important course then databases is an important course"

Propositional Logic.

We form compound propositions by using **logical operators**:
and, or, not, exclusive or, implication(if-then), biconditional(iff)).

A **tautology** is a compound proposition that always evaluates to **true**. e.g.: p v ¬p

A **contradiction** is a compound proposition that always evaluates to **false**.

A **predicate** is a property or description of subjects in the universe of discourse.
In the previous slide, predicates are italicized :
is about databases,
is an important course

Propositional & Predicate Logic - Relational Calculus

Knowing the two propositions

p = "COMP5531 *is about databases*"
r = "databases *is an important course*"

Can we say that COMP5531 is an important course?

Example of a relation as predicates: Assignment (*E#*, *P#*, *H*) expresses the fact that Employee *E#* is assigned to project *P#* for *H* hours

Its value is true if an Employee *E#* is assigned *H* hours to project *P#* *else it is false*

In the database this is expressed by having a tuple in the table for Assignment

⇔ indicates derivable or follows in both directions

Assertion of Universality

$\forall x{:}P(x) \Leftrightarrow \neg\exists x{:}\neg P(x)$

*If everything **is (true)**, there exists nothing that **is not (true)**.*

Denial of Existence

$\forall x{:}\neg P(x) \Leftrightarrow \neg\exists x{:}P(x)$

*If everything **is not(true)**, there exists nothing that **is(true)**.*

Denial of Universality

$\neg\forall x{:}P(x) \Leftrightarrow \exists x{:}\neg P(x)$

*If not everything **is(true)**, there exists something that **is not(true)**.*

Assertion of Existence

$\neg\forall x{:}\neg P(x) \Leftrightarrow \exists x{:}P(x)$

*If not everything **is not(true)**, there exists something that **is(true)**.*

De Morgan’s Laws

It can be shown that the following, called De Morgan’s laws are equivalent:

$P(x) \wedge Q(x) \equiv \neg(\neg P(x) \vee \neg Q(x))$
$P(x) \vee Q(x) \equiv \neg(\neg P(x) \wedge \neg Q(x))$

A generalization of De Morgan's Law involving the $\forall$, $\exists$ quantifiers is obtained as shown in the following.

Assertion of Universality & Assertion of Existence

$\forall x(P(x)) \equiv \neg(\exists x)(\neg P(x))$ and $\exists x(P(x)) \equiv \neg(\forall x)(\neg P(x))$

In formal systems, the acceptable sentences (or formulae) are usually called **well-formed formulae** (wff).

In the wff $(\forall x)(P(x) \,\&\, Q(y))$, where $\forall$ is the universal quantifier (for all), x is **bound** and y is **free**.

Tuple calculus formulae are built from **atoms** of the form:

A_1. x ∈ R where R is a relation and x is a tuple variable.

A_2. x θ y or x θ c where θ ∈{=,≠,<.,,>,≥}, x and y are variables and c is a constant: x, y, c are domain compatible

Formulae are built from atoms using the following rules:

B_1. An atom is a formula.

B_2. If f and g are formulae, then are: ¬f, (f), f ∨ g, f ∧ g, f → g

B_3. If f(x) is a formula, where x is free, then ∃x(f(x)), and ∀x(f(x)) are also formulae; however, x is now bound.

The formula f → g, meaning **if** f **then** g, is equivalent to ¬f ∨ g.

Relational Calculus

Relational calculus is a query system wherein queries are expressed as variables and formulae on these variables. Such a formula describes the properties of the required result relation without specifying the method of evaluating it.

Tuple and domain calculi are collectively referred to as relational calculus.

A query in tuple relational calculus is expressed as a formula:

$$\{t \mid P(t)\}$$

This is the formula that finds all tuples t such that the predicate P is true.

A formula may use a constant to specify a particular value, while a variable is used as a place holder for the values in an expression or procedure.

We can also specify logical connectors such as "not" (or negation; denoted by ¬), "or" (∨), "and" (∧), and "implication" (→), universal (or **for all**; denoted by ∀ and existential (or **for some**; denoted by ∃)

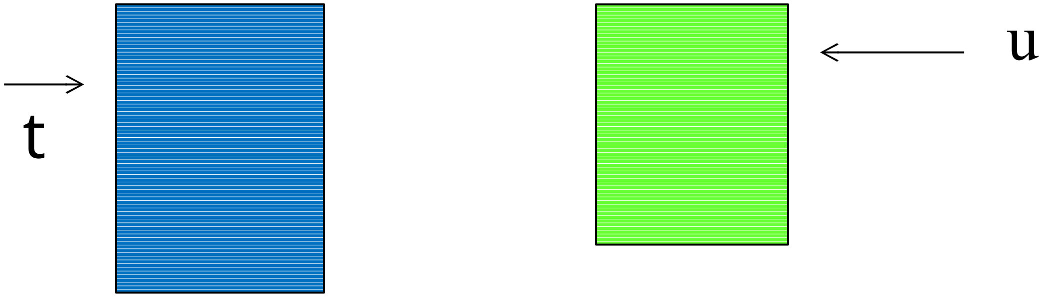
PROJECT (***Project#***, *Project_Name*, *Chief_Architect*)
EMPLOYEE (***Emp#***, *EmpName*)
ASSIGNED_TO (***Project#***, ***Emp#***)

- Obtain the **employee numbers** of employees working on project P1.
- Obtain **employee details** for those employees assigned to project P1
- Get complete details of employees working on **a** Database project.
- Get complete details of employees working on **al**l Database projects.
- List the complete details of employees working on **both** P1 and P2.
- List the complete details of employees working on **either** P1 **or** P2 **or both**.

PROJECT (***Project#***, *Project_Name*, *Chief_Architect*)
EMPLOYEE (***Emp#***, *EmpName*)
ASSIGNED_TO (***Project#***, ***Emp#***)

- Obtain the employee numbers of employees working on project P1.

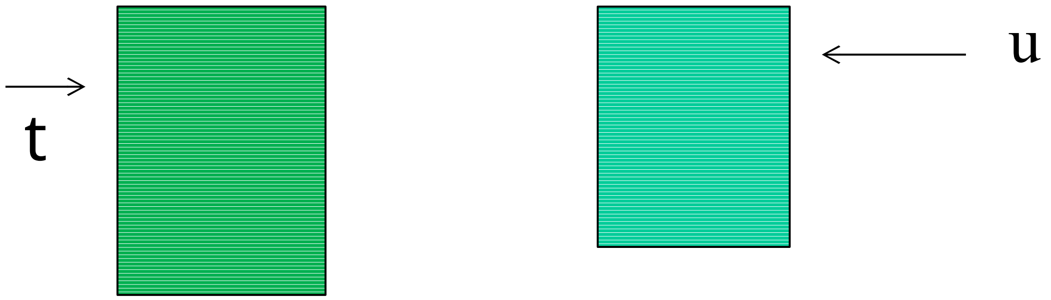
$$\begin{aligned}\{t(\mathit{Emp\#}) \mid \exists u(u \in \mathrm{ASSIGNED\_TO} \land u[\mathit{Project\#}]=\text{'P1'} \\ \land\, t[\mathit{Emp\#}] = u[\mathit{Emp\#}])\ \}\end{aligned}$$



PROJECT (***Project#***, *Project_Name*, *Chief_Architect*)
EMPLOYEE (***Emp#***, *EmpName*)
ASSIGNED_TO (***Project#***, ***Emp#***)

◆ Obtain employee details for those employees assigned to the project P1

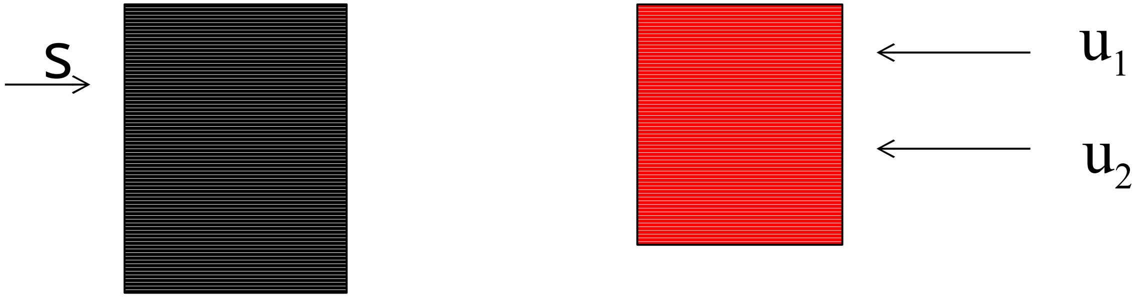
$$\{t \mid t \in \text{employee} \wedge \exists\, u(u \in \text{ASSIGNED\_TO} \wedge \\ u[\mathit{Emp\#}]=t[\mathit{Emp\#}] \wedge \\ u[\mathit{Project\#}] = \text{P1}))\ \}$$



- ❖ List the complete details of employees working on both P1 and P2.

$$\{s \mid s \in \text{employee} \wedge \exists u_1,u_2\ (u_1 \in \text{assigned\_to} \\ \wedge u_2 \in \text{assigned\_to} \wedge u_1[\mathit{Emp\#}] = u_2[\mathit{Emp\#}] \\ \wedge s[\mathit{Emp\#}] = u_1[\mathit{Emp\#}] \wedge u_1[\mathit{Project\#}] = \text{'P1'} \\ \wedge u_2[\mathit{Project\#}] = \text{'P2'})\}$$

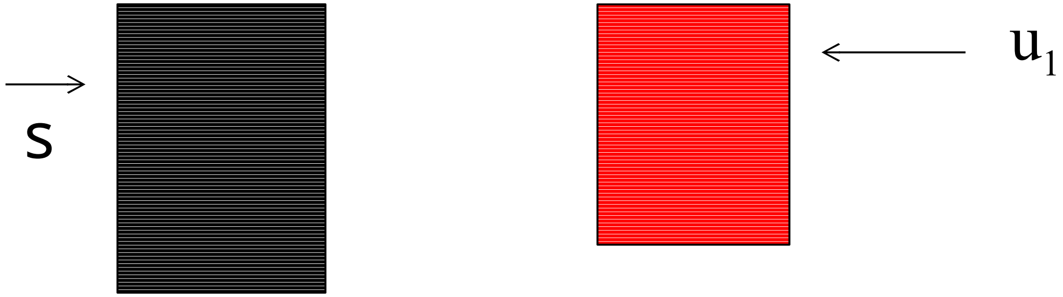
Find s such that s is from employee and there exists tuples u_1,u_2 both from assigned_to such that a number of predicates are being satisfied



List the complete details of employees working on either P1 or P2 or both.

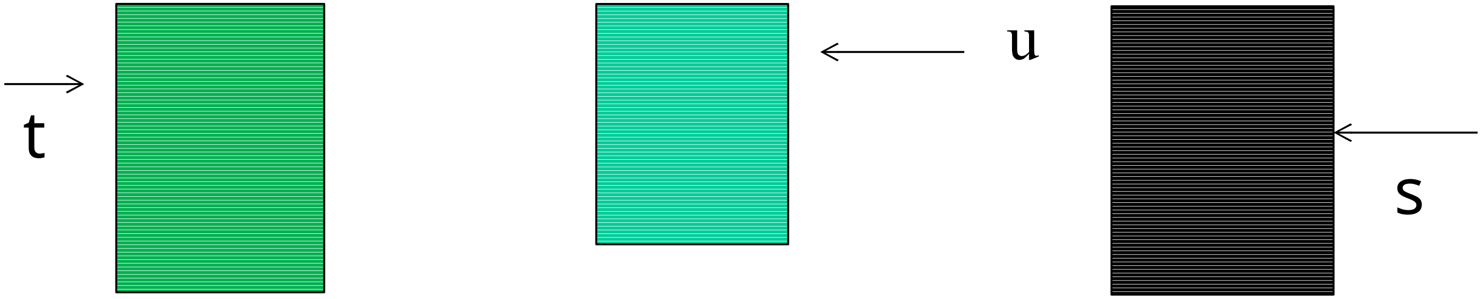
$$\{s \mid s \in \text{employee} \wedge \exists u_{1,} u_2 (u_1 \in \text{assigned\_to} \wedge$$
$$u_2 \in \text{assigned\_to} \wedge ((s[\mathit{Emp\#}] = u_1[\mathit{Emp\#}] \wedge$$
$$u_1[\mathit{Project\#}] = \text{'P1'})$$
$$\vee (s[\mathit{Emp\#}] = u_2[\mathit{Emp\#}] \wedge u_2[\mathit{Project\#}] = \text{'P2'})))\}$$

$$\{s \mid s \in \text{employee} \wedge \exists u_1 (u_1 \in \text{assigned\_to}$$
$$\wedge\ s[\mathit{Emp\#}] = u_1[\mathit{Emp\#}] \wedge (u_1[\mathit{Project\#}] = \text{'P1'}$$
$$\text{} u_1[\mathit{Project\#}] = \text{'P2'}))\}$$



- Get complete details of employees working on a Database project.

$$
\begin{aligned}
\{s \mid\ & s \in \text{employee} \\
& \wedge \exists u,t(t \in \text{project} \wedge t[\mathit{Project\_Name}] = \text{'Database'} \\
& \wedge u \in \text{assigned\_to} \wedge u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
& \wedge s[\mathit{Emp\#}] = u[\mathit{Emp\#}])\}
\end{aligned}
$$



The universe of discourse for a particular branch of mathematics is a set that contains everything of interest for that subject.

If P and Q are formulas, then "if P then Q" or
"P implies Q" is written $P \Rightarrow Q$, using the conditional symbol, $\Rightarrow$.

Bi-conditional, written $\Leftrightarrow$, corresponds to the phrase
"if and only if" or "iff" for short.

The denial or negation of of $P \Rightarrow Q$ can be expressed as:

$\neg(P \Rightarrow Q) \Leftrightarrow \neg(\neg P \vee Q)$ or
$\neg(P \Rightarrow Q) \Leftrightarrow \neg\neg P) \wedge (\neg Q)$
$\neg(P \Rightarrow Q) \Leftrightarrow P \wedge \neg Q$

De Morgan's laws for quantifier are expresses usually in the form:

$\neg\forall x P(x) \Leftrightarrow \exists x \neg P(x)$

$\neg\exists x P(x) \Leftrightarrow \forall x \neg P(x)$ which could be re-written as:

$\forall x \neg P(x) \Leftrightarrow \neg\exists x P(x)$

Get details of employees working on all Database projects.

$\{s \mid s \in employee$

$\wedge\ \forall t(t \in project \wedge t[\mathit{Project\_Name}] = \text{'Database'}$

$\rightarrow \exists u(u \in assigned\_to \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}]$

$\wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}]))\}$

Writting the predicate as

$s \in employee \wedge\ \forall t\ (\neg[(\neg(P(t) \rightarrow\ \exists u Q(u,t))])$

Now we use: $\forall x \neg P(x) \Leftrightarrow \neg \exists x P(x)$ and re-write the above predicate as:

$s \in employee \wedge \neg \exists t(\neg(P(t) \rightarrow \exists u Q(u,t)))$

Substituting $f \rightarrow g$ by its equivalent form $\neg f \vee g$:

We get: $s \in employee \wedge \neg \exists t(\neg(\neg P(t) \vee \exists u Q(u,t)))$

Now move the negation in and stop it just **after** the $\vee$

$s \in employee \wedge \neg \exists t(\neg(\neg P(t) \vee \exists u Q(u,t)))$

$s \in employee \wedge \neg \exists t(P(t) \wedge \neg \exists u Q(u,t)))$

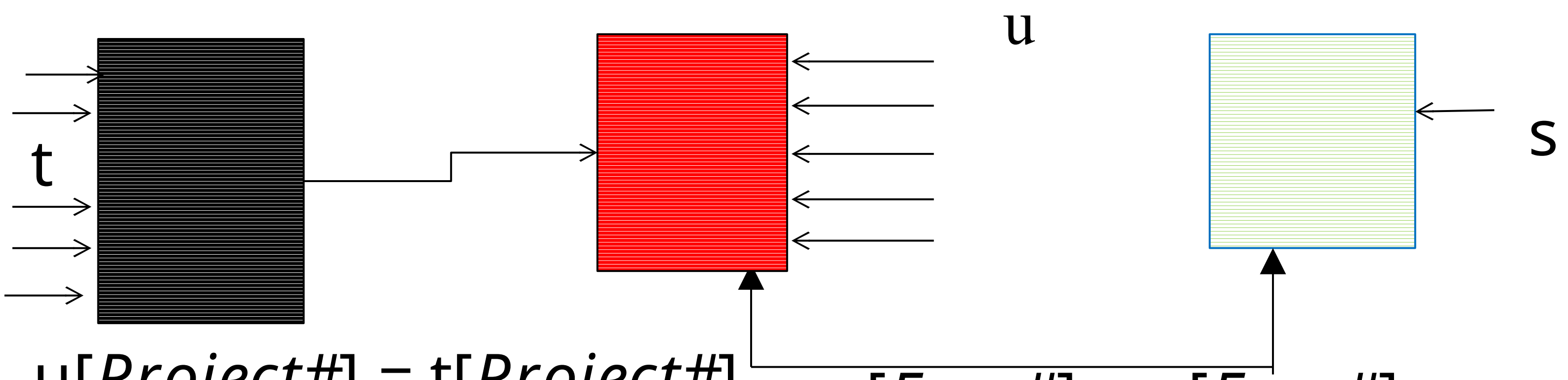
{s | s ∈ employee
∧ ¬ ∃t(t ∈ project ∧ t[*Project_Name*] = 'Database'
∧ ¬ ∃u(u ∈ assigned_to ∧ u[*Project#*] = t[*Project#*]
∧ s[*Emp#*] = u[*Emp#*]))}

- Get details of employees working on all Database projects.

$$\begin{aligned}
&\{s \mid s \in \text{employee} \\
&\qquad \wedge\ \forall t(t \in \text{project} \wedge t[\mathit{Project\_Name}] = \text{'Database'} \\
&\qquad \rightarrow \exists u(u \in \text{assigned\_to} \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
&\qquad \wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}]))\}
\end{aligned}$$

replacing f → g by its equivalent form $\neg f \vee g$:

$$\begin{aligned}
&\{s \mid s \in \text{employee} \\
&\qquad \wedge\ \forall t(t \notin \text{project} \vee t[\mathit{Project\_Name}] \neq \text{'Database'} \\
&\qquad \vee\ \exists u(u \in \text{assigned\_to} \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
&\qquad \wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}]\ ))\}
\end{aligned}$$



$u[\mathit{Project\#}] = t[\mathit{Project\#}]$ $\quad s[\mathit{Emp\#}] = u[\mathit{Emp\#}]$

```
mysql> select * from Assign;
+-----+-------+   mysql> select * from Project;
| Pno | Eno   |      +--------+----------+---------+
+-----+-------+      | ProjNo | Pname    | Pleader |
| 353 | 10000 |      +--------+----------+---------+
| 354 | 10000 |      |    353 | Database |   10000 |
| 534 | 10001 |      |    354 | Database |   10000 |
| 353 | 10002 |      |    534 | OS       |   10005 |
| 354 | 10003 |      |    574 | VOIP     |   10005 |
| 534 | 10003 |      +--------+----------+---------+
| 354 | 10004 |      4 rows in set (0.00 sec)
| 534 | 10005 |
| 574 | 10005 |
+-----+-------+mysql> select * from Employee;
+-------+----------+-------------+------------+------------+
| EmpNo | Ename    | Address     | Phone      | DOB        |
+-------+----------+-------------+------------+------------+
| 10000 | James    | Montreal    | 5144445555 | 1965-10-21 |
| 10001 | Piere    | Laval       | 5144555445 | 1956-10-12 |
| 10002 | Nathalie | Brossard    | 5147454555 | 1976-04-01 |
| 10003 | Mary     | Dorval      | 5145544455 | 1965-10-21 |
| 10004 | Sabrina  | St. Laurent | 5144445555 | 1987-01-31 |
| 10005 | Ma       | Montreal    | 5144454555 | 1964-02-29 |
+-------+----------+-------------+------------+------------+
```

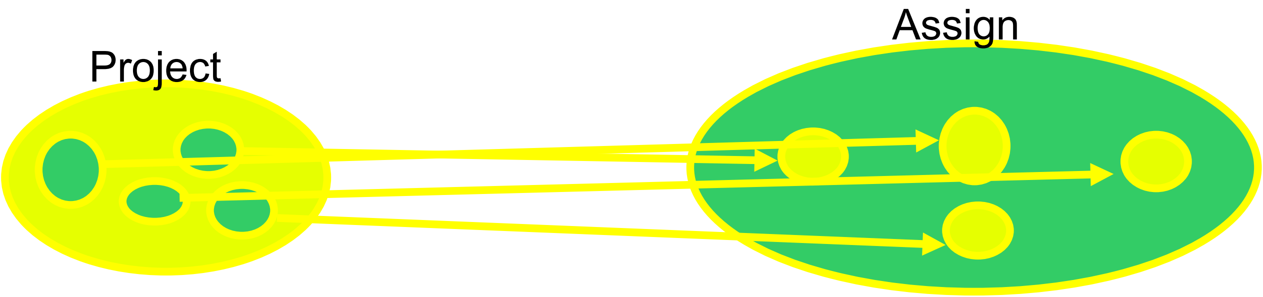
Get details of employees working on **all** Database projects.

$\{s \mid s \in employee$

$\wedge\ \forall t(\ (t \in project \wedge t[\mathit{Project\_Name}] = \text{'Database‘})$

$\rightarrow (\ \exists u(u \in assigned\_to \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}]$

$\wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}]\ )\ )\ )\ \}$



replacing (f) → (g) by its equivalent form (¬f) ˅ (g) :

$\{s \mid s \in employee$

$\wedge\ \forall t(\ (t \notin project \vee t[\mathit{Project\_Name}] \neq \text{'Database‘}\ )$

$\vee\ (\exists u(u \in assigned\_to \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}]$

$\wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}]\ )\ )\ )$

$$
\begin{aligned}
&\{s \mid s \in \text{employee} \\
&\qquad \wedge\ \forall t(t \notin \text{project}\ {}^{\vee}\ t[\mathit{Project\_Name}] \neq \text{'Database'} \\
&\qquad \vee\ \exists u(u \in \text{assigned\_to} \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
&\qquad \wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}]))\}
\end{aligned}
$$

Now use Assertion of Universality: $\forall x(P(x)) \equiv \neg(\exists x)(\neg P(x))$

$$
\begin{aligned}
&\{s \mid s \in \text{employee} \\
&\qquad \wedge\ \neg(\exists t)(\neg(t \notin \text{project} \vee t[\mathit{Project\_Name}] \neq \text{'Database'} \\
&\qquad \vee\ \exists u(u \in \text{assigned\_to} \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
&\qquad \wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}])))\}
\end{aligned}
$$

$$
\begin{aligned}
&\{s \mid s \in \text{employee} \\
&\qquad \wedge\ \neg(\exists t)((t \in \text{project} \wedge t[\mathit{Project\_Name}] = \text{'Database'} \\
&\qquad \wedge\ \neg\exists u(u \in \text{assigned\_to} \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
&\qquad \wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}])))\}
\end{aligned}
$$

not
exists

```
select *
from Employee  e
where not exists
     (select *
      from Project t
      where PName='Database'
      and not exists
                (select * from Assign u
                       where u.Pno = t.ProjNo  and
                        e.EmpNo = u.Eno));
```

```
+-------+-------+----------+------------+------------+
| EmpNo | Ename | Address  | Phone      | DOB        |
+-------+-------+----------+------------+------------+
| 10000 | James | Montreal | 5144445555 | 1965-10-21 |
+-------+-------+----------+------------+------------+
select *
from Employee  e
where not exists
     (select ProjNo
     from Project t
      where PName='Database' and
      ProjNo NOT IN
         (select a.Pno
           from Assign a
           where a.Eno = e.EmpNo) );
```

```
+-------+-------+----------+------------+------------+
| EmpNo | Ename | Address  | Phone      | DOB        |
+-------+-------+----------+------------+------------+
| 10000 | James | Montreal | 5144445555 | 1965-10-21 |
+-------+-------+----------+------------+------------+
insert Employee values(10006, 'John', 'Ndg', 5144455555, '1988-04-01');
insert Assign values (353, 10006),(354,  10006),(534, 10006),(574,10006);
mysql> select * from Employee s
where not exists (select * from Project t
                        where not exists(select * from Assign u
                                    where u.Pno = t.ProjNo  and
                                     s.EmpNo = u.Eno));
 mysql> select * from Employee  e
 where not exists
      (select *
       from Project
       where ProjNo  NOT IN
                 (select distinct a.Pno
                  from Assign a
                 where a.Eno = e.EmpNo) );
+-------+-------+----------+------------+------------+
| EmpNo | Ename | Address  | Phone      | DOB        |
+-------+-------+----------+------------+------------+
| 10006 | John  | Ndg      | 5144455555 | 1988-04-01 |
+-------+-------+----------+------------+------------+
mysql> update Employee set DOB = '1988-01-01' where
EmpNo=10006;
mysql> update Employee set DOB = DATE_ADD(DOB,
INTERVAL 365 DAY) where EmpNo=10006;
```

```
mysql> select * from Employee where EmpNo=10006;
+-------+-------+---------+------------+------------+
| EmpNo | Ename | Address | Phone      | DOB        |
+-------+-------+---------+------------+------------+
| 10006 | John  | Ndg     | 5144455555 | 1988-12-31 |
+-------+-------+---------+------------+------------+
1 row in set (0.00 sec)
```

```
mysql>update Employee set DOB = DATE_SUB(DOB,
INTERVAL 365 DAY) where EmpNo=10006;
mysql> select * from Employee where EmpNo=10006;
```

```
+-------+-------+---------+------------+------------+
| EmpNo | Ename | Address | Phone      | DOB        |
+-------+-------+---------+------------+------------+
| 10006 | John  | Ndg     | 5144455555 | 1988-01-01 |
+-------+-------+---------+------------+------------+
1 row in set (0.02 sec)
```

Example: Get the employee numbers of employees, other than employee 107, who work on at least all those projects that employee 107 works on"

$$
\begin{aligned}
\{\ & t[\mathit{Emp\#}] \mid t \in \text{assigned\_to} \wedge \\
& \quad \forall u_1(u_1 \in \text{assigned\_to} \wedge u_1[\mathit{Emp\#}] = 107 \\
& \quad \rightarrow \exists u_2(u_2 \in \text{assigned\_to} \wedge u_2[\mathit{Emp\#}] \neq 107 \\
& \quad \wedge\ u_1[\mathit{Project\#}] = u_2[\mathit{Project\#}] \wedge t[\mathit{Emp\#}] = u_2[\mathit{Emp\#}]))\}
\end{aligned}
$$

Alternately we can write this query without the logical implication by substituting its equivalent form $\neg f \vee g$:

$$
\begin{aligned}
\{\ & t[\mathit{Emp\#}] \mid t \in \text{assigned\_to} \wedge \\
& \quad \forall u_1(u_1 \notin \text{assigned\_to} \vee u_1[\mathit{Emp\#}] \neq 107 \\
& \quad \vee\ \exists u_2(u_2 \in \text{assigned\_to} \wedge u_2[\mathit{Emp\#}] \neq 107 \\
& \quad \wedge\ u_1[\mathit{Project\#}] = u_2[\mathit{Project\#}] \wedge t[\mathit{Emp\#}] = u_2[\mathit{Emp\#}]))\}
\end{aligned}
$$

To avoid procedural operation, such as projection, in a calculus query, we could define t to be on the relation scheme (*Emp#*) and rewrite this query expression as:

$$
\begin{aligned}
\{\, & t(\mathit{Emp\#}) \mid \\
& \forall u_1(u_1 \notin \text{assigned\_to} \vee u_1[\mathit{Emp\#}] \neq 107 \\
& \vee \exists u_2(u_2 \in \text{assigned\_to} \wedge u_2[\mathit{Emp\#}] \neq 107 \\
& \wedge u_1[\mathit{Project\#}] = u_2[\mathit{Project\#}] \wedge t[\mathit{Emp\#}] = u_2[\mathit{Emp\#}]))\}
\end{aligned}
$$

Example: Get employee numbers of employees who do not work on project P2.

$$
\begin{aligned}
&\{ \text{t}[\mathit{Emp\#}] \mid \text{t} \in \text{assigned\_to} \land \\
&\qquad \neg\exists \text{u}(\text{u} \in \text{assigned\_to} \land \text{u}[\mathit{Project\#}] = \text{P2} \\
&\qquad \land\ \text{t}[\mathit{Emp\#}] = \text{u}[\mathit{Emp\#}])\}
\end{aligned}
$$

Alternatively, we can express this query in the following equivalent form:

$$
\begin{aligned}
&\{ \text{t}[\mathit{Emp\#}] \mid \text{t} \in \text{assigned\_to} \land \\
&\qquad \forall \text{u}(\text{u} \notin \text{assigned\_to} \lor \text{t}[\mathit{Emp\#}] \neq \text{u}[\mathit{Emp\#}] \\
&\qquad\qquad \lor\ \text{u}[\mathit{Project\#}] \neq \text{P2})\}
\end{aligned}
$$

Example: Compile a list of employee numbers of employees who work on all projects.

$$
\begin{aligned}
\{\ & t[\mathit{Emp\#}] \mid t \in \text{assigned\_to} \wedge \\
& \quad \forall p(p \in \text{PROJECT} \rightarrow \exists u(\ u \in \text{assigned\_to} \\
& \quad \wedge\ p[\mathit{Project\#}] = u[\mathit{Project\#}] \\
& \quad \wedge\ t[\mathit{Emp\#}] = u[\mathit{Emp\#}]))\}
\end{aligned}
$$

The above can be rewritten as:

$$
\begin{aligned}
\{\ & t[\mathit{Emp\#}] \mid t \in \text{assigned\_to} \wedge \\
& \quad \forall\ p(p \notin \text{PROJECT} \vee \exists u(\ u \in \text{assigned\_to} \\
& \quad \wedge\ p[\mathit{Project\#}] = u[\mathit{Project\#}] \\
& \quad \wedge\ t[\mathit{Emp\#}] = u[\mathit{Emp\#}]))\}
\end{aligned}
$$

Example: Get employee numbers of employees, not including employee 107, who work on at least one project that employee 107 works on.

$$
\begin{aligned}
\{\, & t[\mathit{Emp\#}] \mid t \in \text{assigned\_to} \wedge \\
& \quad \exists s,u\, (s \in \text{assigned\_to} \wedge u \in \text{assigned\_to} \\
& \quad \wedge s[\mathit{Project\#}] = u[\mathit{Project\#}] \\
& \quad \wedge s[\mathit{Emp\#}] = 107 \\
& \quad \wedge t[\mathit{Emp\#}] \neq 107 \\
& \quad \wedge t[\mathit{Emp\#}] = u[\mathit{Emp\#}])\}
\end{aligned}
$$

Consider the division operation on the two relations, P(**P**) and Q(**Q**), where **Q** ⊆ **P**:

$R = P \div Q$

$R = \{t \mid t \in P[\mathbf{P}\text{-}\mathbf{Q}] \wedge \forall s(s \in Q \wedge (t \mid\mid s \in P)\}$

$R = \{t \mid t \in P[\mathbf{P}\text{-}\mathbf{Q}] \wedge \forall s(s \in Q \rightarrow \exists u(u \in P \wedge u[\mathbf{Q}]=s \wedge u[\mathbf{P}\text{-}\mathbf{Q}]=t[\mathbf{P}\text{-}\mathbf{Q}])))\}$

$R = P \div Q = \Pi_{\mathbf{P}\text{-}\mathbf{Q}}(P) - \Pi_{\mathbf{P}\text{-}\mathbf{Q}}((\Pi_{\mathbf{P}\text{-}\mathbf{Q}}(P) \times Q) - P)$

$\Pi_{P-Q}(P)$

A
a_1
a_2
a_3
a_4
a_5

Q

B
b_1
b_2

$\Pi_{P-Q}(P) \times Q$

A	B
a_1	b_1
a_2	b_1
a_3	b_1
a_4	b_1
a_5	b_1
a_1	b_2
a_2	b_2
a_3	b_2
a_4	b_2
a_5	b_2

$\Pi_{P-Q}(P) \times Q - P$

A	B
a_4	b_1
a_2	b_2
a_3	b_2

$\Pi_{P-Q}(\Pi_{P-Q}(P) \times Q - P)$

A
a_4
a_2
a_3

$\Pi_{P-Q}(P) -$

$\Pi_{P-Q}(\Pi_{P-Q}(P) \times Q - P)$

A
a_1
a_5

A domain calculus expression is of the form

$$\{ X \mid f(X) \}$$

where f is a formula on X, and X represents a set of domain variables.

A_1. $X \in \mathbf{R}$

A_2. $x\ \theta\ y$ or $x\ \theta\ c$

where θ is one of the comparison operators x and y are domain compatible variables, and c is a domain compatible constant.

B_1. An atom is a formula.

B_2. If f and g are formulae, then $\neg f$, (f), $f \wedge g$, $f \wedge g$, $f \rightarrow g$ are also formulae.

B_3. If f(X) is a formula where X is free, then $\exists X(f(X))$, and $\forall X(f(X))$ are also formulae

PROJECT(***Project#***,*Project_Name*,*Chief_Architect*)
EMPLOYEE(***Emp#***, *EmpName*)
ASSIGNED_TO (***Project#***, ***Emp#***)

Get employee numbers for employees working on project
number P1

$\{e \mid \exists p\,(\langle e, p\rangle \in \text{assigned\_to} \land p = \text{P1})\,\}$

In this can, we can drop the quantifier and simplify

the query as:

$\{e \mid \langle e, p\rangle \in \text{assigned\_to} \land p = \text{P1}\}$

Get employee details such that the employee is assigned to the project P1

$$\{<e_1, m> \mid \exists e_2 (<p, e_2> \in \text{assigned\_to}$$
$$\wedge <e_1, m> \in \text{employee}) \wedge p = \text{P1} \wedge e_1 = e_2)\}$$

Compile the details of employees working on a Database project.

$$\{e,m \mid \exists p_1,e_1,p_2,n_2 (<p_1,e_1> \in \text{assigned\_to}$$
$$\wedge <e,m> \in \text{employee}$$
$$\wedge <p_2,n_2,c_2> \in \text{project}$$
$$\wedge e_1 = e \wedge p_1 = p_2 \wedge n_2 = \text{Database})\}$$

Compile the details of employees working on both P1 and P2.

$$
\begin{aligned}
\{e,m \mid \exists p_1,e_1,p_2,e_2\ (&<e,m> \in \text{employee} \\
&\wedge <p_1,e_1> \in \text{assigned\_to} \\
&\wedge <p_2,e_2> \in \text{assigned\_to} \\
&\wedge e = e_1 \wedge e = e_2 \\
&\wedge p_1 = \text{'P1'} \wedge p_2 = \text{'P2'})\}
\end{aligned}
$$

Obtain the employee numbers of employees, other than the employee 107, who work on at least all those projects that employee 107 works on.

$$\{e \mid <p,e> \in assigned\_to\ \forall\ p_1,e_1(\\ <p_1,e_1> \in assigned\_to\ \wedge e_1 = 107 \\ \rightarrow (\exists p_2,e_2(<p_2,e_2> \in assigned\_to \\ \wedge e_2 \neq 107 \wedge\ p_1 = p_2 \wedge e = e_2))\}$$

An equivalent form of this query

$$\{e \mid <p,e> \in assigned\_to \wedge \\ \forall p_1,e_1(<p_1,e_1> \notin assigned\_to\ \vee e_1 \neq 107 \\ \vee (\exists p_2,e_2(<p_2,e_2> \in assigned\_to \\ \wedge e_2 \neq 107 \wedge p_1 = p_2 \wedge e = e_2))\}$$

Get employee numbers of employees who do not work on the P2 project.

$$\{e \mid \exists p\,(\langle p,e\rangle \in assigned\_to \\ \wedge\ \forall\, p_1,e_1\,(\langle p_1,e_1\rangle \notin assigned\_to \\ \vee\ p_1 \neq P2 \vee e_1 \neq e))\}$$

What are the employee numbers of employees who work on all projects?"

$$\{e \mid \exists p\,(\ \langle p,e\rangle \in assigned\_to \\ \wedge\ \forall\, p_1(\langle p_1,n_1,c_1\rangle \in project \\ \rightarrow \langle p_1,e\rangle \in assigned\_to\ ))\}$$

Acquire the employee numbers of employees, other than employee 107, who work on at least one project that employee 107 works on.

$$\begin{aligned}\{e \mid \exists\, p,p_1,e_1,p_2,e_2(&<p,e> \in \text{assigned\_to} \\ &\wedge <p_1,e_1> \in \text{assigned\_to} \\ &\wedge <p_2,e_2> \in \text{assigned\_to} \\ &\wedge e_2 \neq 107 \wedge p_1 = p_2 \wedge e_1 = 107 \wedge e = e_2)\}\end{aligned}$$

The following is covered in predicate logic discussions: Here Q(x) is any predicate of variable x and $\equiv$ means Logically equivalent

Negating a proposition such as $\forall x Q(x)$ requires negating the predicate and changing the quantifier from the Universal to the existential

Thus we can replace a negated universal quantifier with a predicate Q(x) as follows to its logically equivalent form

$\neg[\forall x Q(x)] \equiv \exists x[\neg Q(x)]$ or $\exists x[\neg Q(x)] \equiv \neg[\forall x Q(x)]$

Similarly, we can negate a proposition with an existential quantifier as follows:

$\neg[\exists x Q(x)] \equiv \forall x[\neg Q(x)]$ or $\forall x[\neg Q(x)] \equiv \neg[\exists x Q(x)]$

Using the last formula from the previous slide :

$\neg[\exists x Q(x)] \equiv \forall x[\neg Q(x)]$

If we reverse the sides and move the negation inside:

$\forall x(\neg Q(x)) \equiv \neg(\exists x)(\neg Q(x))$

Now if we substitute, in the above:
$P(x)$ for $\neg Q(x)$ and $\neg P(x)$ for $\neg \neg Q(x)$ i.e., $\neg P(x)$ for $Q(x)$

We get:
$\forall x(P(x)) \equiv \neg[(\exists x)(\neg Q(x))]$ or
$\forall x(P(x)) \equiv \neg[(\exists x) (\neg\neg P(x))]$ or
$\forall x(P(x)) \equiv \neg[(\exists x) (P(x))]$

It is the last form that we have used!!

Get details of employees working on **al**l Database projects .

$$
\begin{aligned}
&\{s \mid s \in \text{employee} \;\wedge \\
&\;\neg\exists t((t \in \text{project} \wedge \\
&\quad \wedge\; t[\mathit{Project\_Name}] = \text{'Database'} \\
&\quad \wedge\;\; \neg\exists u(u \in \text{assigned\_to} \wedge \\
&\;\; u[\mathit{Project\#}] = t[\mathit{Project\#}] \wedge s[\mathit{Emp\#}] = u[\mathit{Emp\#}])))\}
\end{aligned}
$$

```
select   distinct  a.empno
   from      assigned_to a, project p
   where     not exists
            (select *
             from project p
             where p.projname = 'database' and
                 not exists
                  (select *
                   from assigned_to a1
                   where a1.projno = p.projno and
                       a1.empno = a.empno));
```

Get details of employees working on **all** Database projects and **only** on database projects.

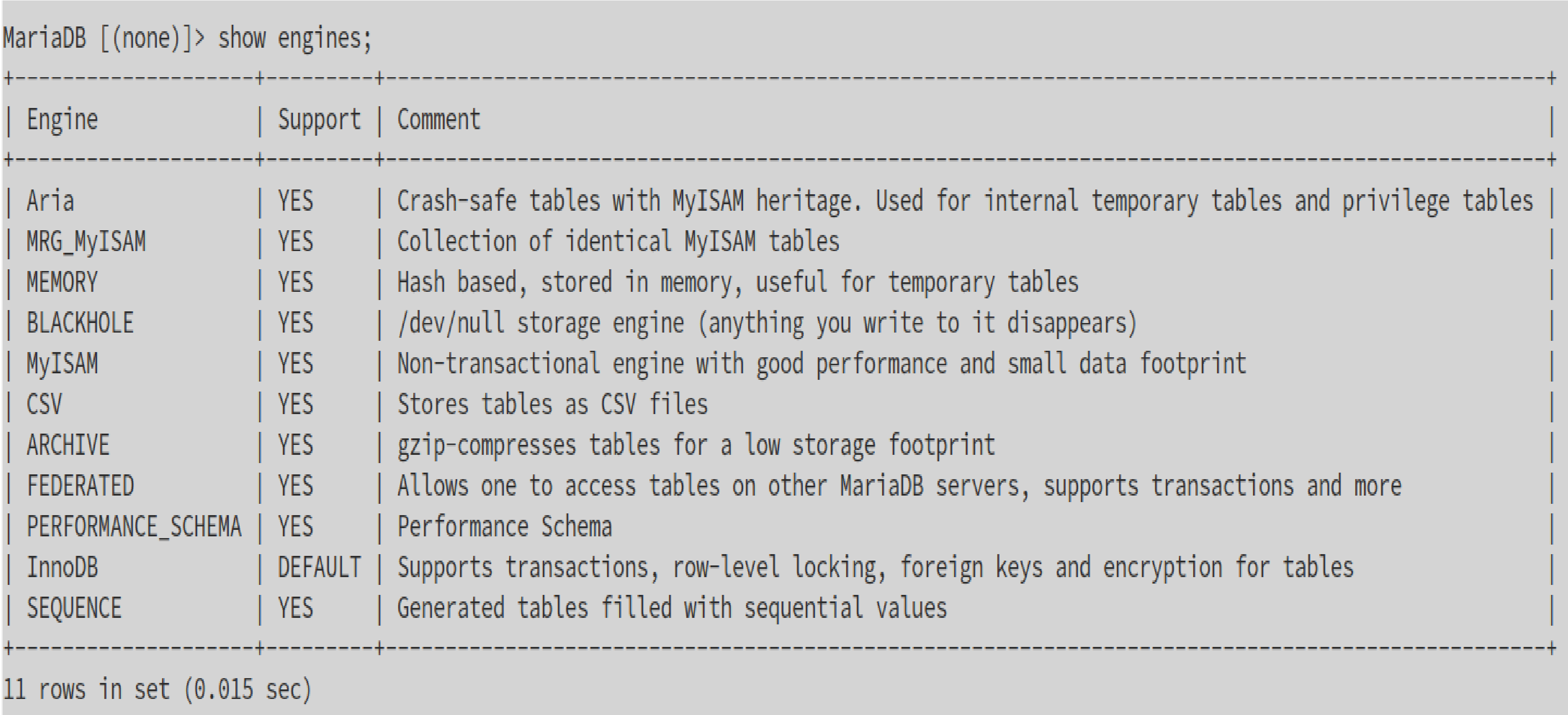
$\{s \mid s \in \text{employee} \ \wedge$
$\neg\exists t((t \in \text{project} \wedge$
$\wedge\ t[\mathit{Project\_Name}] = \text{'Database'}$
$\wedge\ \neg\exists u(u \in \text{assigned\_to} \wedge$
$u[\mathit{Project\#}] = t[\mathit{Project\#}] \wedge s[\mathit{Emp\#}] = u[\mathit{Emp\#}])))$
$\wedge\ (\neg\exists t1(t1 \in \text{project} \wedge t1[\mathit{Project\_Name}] \neq \text{'Database'}$
$\wedge\ \exists u1(u1 \in \text{assigned\_to} \wedge$
$u1[\mathit{Project\#}] = t1[\mathit{Project\#}] \wedge s[\mathit{Emp\#}] = u1[\mathit{Emp\#}]))))\}$

```
select   distinct  a.empno
    from      assigned_to a, project p
    where     not exists
               (select *
                from project p
                where p.projname = 'database' and
                     not exists
                      (select *
                       from assigned_to a1
                       where a1.projno = p.projno and
                            a1.empno = a.empno))
             and not exists ( select *
                             from project p1
                             where p1.projname <> 'database' and
                     exists (select *
                       from assigned_to a2
                       where a2.projno = p1.projno and
                            a2.empno = a.empno));
```

```
mysql> show engines;
+------------+---------+--------------------------------------------+
| Engine      | Support | Comment                                     |
+------------+---------+--------------------------------------------+
| MyISAM      | DEFAULT | Default engine as of MySQL 3.23             |
|             |         | with great performance                      |
| MEMORY      | YES     | Hash based, stored in memory,               |
|             |         |  useful for temporary tables                |
| InnoDB      | YES     | Supports transactions, row-level            |
|             |         |  locking, and foreign keys                  |
| BerkeleyDB  | NO      | Supports transactions and                   |
|             |         |  page-level locking                         |
| BLACKHOLE   | NO      | /dev/null storage engine (anything          |
|             |         |   you write to it disappears)               |
| EXAMPLE     | NO      | Example storage engine                      |
| ARCHIVE     | NO      | Archive storage engine                      |
| CSV         | NO      | CSV storage engine                          |
| ndbcluster  | NO      | Clustered, fault-tolerant,                  |
| ndbcluster  | NO      |  memory-based tables                        |
| FEDERATED   | NO      | Federated MySQL storage engine              |
| MRG_MYISAM  | YES     | Collection of identical MyISAM tables       |
| ISAM        | NO      | Obsolete storage engine                     |
+------------+---------+---------------------------------------------+
12 rows in set (0.02 sec)
```

Evolving database systems

```
MariaDB [mysql]> show engines;

+--------------------+---------+----------------------------------------------+
| Engine             | Support | Comment                                      |
+--------------------+---------+----------------------------------------------+
| MRG_MyISAM         | YES     | Collection of identical MyISAM tables        |
| MyISAM             | YES     | MyISAM storage engine                        |
| MEMORY             | YES     | Hash based, stored in memory,                |
|                    |         |  useful for temporary tables                 |
| CSV                | YES     | CSV storage engine                           |
| Aria               | YES     | Crash-safe tables with MyISAM heritage       |
| InnoDB             | DEFAULT | Percona-XtraDB, Supports transactions, row-level|
|                    |         | locking, foreign keys and encryption for tables |
| SEQUENCE           | YES     | Generated tables filled with sequential values  |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                           |
+--------------------+---------+----------------------------------------------+
```



```
MariaDB [(none)]> show engines;
+--------------------+---------+-------------------------------------------------------------------------------------------------+
| Engine             | Support | Comment                                                                                         |
+--------------------+---------+-------------------------------------------------------------------------------------------------+
| Aria               | YES     | Crash-safe tables with MyISAM heritage. Used for internal temporary tables and privilege tables |
| MRG_MyISAM         | YES     | Collection of identical MyISAM tables                                                           |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables                                       |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears)                                  |
| MyISAM             | YES     | Non-transactional engine with good performance and small data footprint                         |
| CSV                | YES     | Stores tables as CSV files                                                                      |
| ARCHIVE            | YES     | gzip-compresses tables for a low storage footprint                                              |
| FEDERATED          | YES     | Allows one to access tables on other MariaDB servers, supports transactions and more            |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                                                              |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, foreign keys and encryption for tables                |
| SEQUENCE           | YES     | Generated tables filled with sequential values                                                  |
+--------------------+---------+-------------------------------------------------------------------------------------------------+
11 rows in set (0.015 sec)
```

Course Notes
Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

SQL – II

Bipin C. DESAI

This set of slides is extensive with many examples using Oracle and MariaDB/MySQL.
These example should be tried out for better understanding.
We will not go through them in details: this is left as exercises!

Here are the main SQL topics – their syntax etc.- DBMS dependent!

Data Types	-3
SQL statements	-5
Group/having	-14
Functions	-25
Views, Null and operations on null	-36
Joins – cross. inner, natural, outer,	-48
Implementation of joins	-67
Representing relationships, key constraints	-70
Complex relationships	-76
Constraints,triggers(row/statement, before/after);	-70
mutating triggers	-133
Aggregation	-160

DATA TYPES

Typical data types supported are:

integer, decimal, real or float, binary, blob

characters (fixed and variable length), bits, date,

int or **integer; tinyint** (1 byte), **smallin**t (2 bytes),

mediumint (3 bytes), **int** (4 bytes) **bigint** (8 bytes)

real or **float**

decimal(*n*, *d*) -- **numeric**(n, d) e.g. **decimal**(6, 2)

char(n)/**bit**(B) fixed length character/bit string, padded

varchar(n) / **bit varying**(n) variable-length strings up to **n** characters

tinyblob (2^8 -1 bytes), **blob** (2^{16} -1 bytes),

mediumblob (2^{24} -1 bytes), **longblog** (2^{32} -1 bytes)

Note: In the following , we have used MariaDB/MySQL or Oracle

-The prompt for Oracle is: SQL>

-The prompt for MariaDB/MySQL is followed by database name

Oracle also uses **varchar2**(n); it's truly varying length

varchar2 is not yet supported in MySQL/Mariadb!

Times:**SQL2** form is **TIME** 'hh:mm:ss[.ss...]'

Dates: **SQL2** format is **DATE** 'yyyy-mm-dd' (m =0 or 1)

supported in MySQL/Mariadb

Oracle's default dates format is 'dd-mon-yy'

Example: **create table** BDays(name char(25), d DATE);

insert into BDays **values** ("Martha Smith", '18-nov-1962');

Oracle function **to_date** converts a date value into default format, e.g.,

insert into BDays **values**("Martha Smith",

to_date('1962-nov-18', 'yyyy-mm-dd'));

SQL is *case insensitive*

However, the case is significant for ***strings*** and could be for names of tables and columns

Two strings s_1 and $\mathbf{s}_2$ are equal if

- they have the same sequence of characters and

- the same case

The strings are compared alphabetically

’fodder’ < ’folder’

’**bat**’ < ’**bat**man’

Note: some DBMS are case sensitive for the names of tables and columns: all DBMS are case sensitive for strings

string **LIKE** *pattern*

Ordinary characters in *pattern* match only ordinary characters in *string*

The special character **%** in *pattern*, matches any sequence of zero or more characters in *string*

The special character **_** in *pattern*, matches any **one** character in *string*

Find all students with “de’” in their name

select Name, Dept

from students

where Name **like** ’%de” %’ **;**

Note: an apostrophe in a string represented by two apostrophes ” **without any intervening spaces**

Expressing special characters _ and % in a string by an using a preceding escape character.

SQL allows any character to be used as an escape character with the *escape* keyword

string **LIKE** ’x_%x%’ **ESCAPE** ’x’

Here **x** is the escape character

The sequence ’x_ and x% is taken to be a single _ and **%**

This pattern matches any string that begins with _ and ends with **%**

We can apply the 3 most common set operations **union, intersection, except(difference)** to relations **R** and **S,** provided the relations are *compatible*.

If two SQL queries produce compatible relations as their result, then we may combine these queries using: **union, intersection, except**

The SQL implementation of **union**, **intersection**, and **except** operation normally eliminate duplicates; the modifier ALL is used to keep the duplicates:

R UNION ALL *S*, $|t \in R| = n,\ |t \in S| = m,\ |t \in R \cup S| \leq n+m$

R INTERSECT ALL *S*, $|t \in R| = n,\ |t \in S| = m\ |t \in R \cap S| = min(n,m)$

R EXCEPT ALL *S* , $|t \in R| = n,\ |t \in S| = m\ |t \in R - S| = max(0,n-m)$

NOTE: in many of the SQL examples, to save space on slides, we are not including constraints such a primary key which are usually evident

Relation schemas:

Faculty (Name, Dept, Position, salary, gender)

Student (Name, Dept, Major, sex)

Query:

Give the names and the departments of students and professors.

```
(SELECT Name, Dept
FROM Faculty)
          UNION
(SELECT Name, Dept
FROM Student);
```

```
create table Faculty (  Name varchar2(30),
 Dept varchar2(20),   Position varchar2(20),
 salary  dec(9,2),   gender char(1));
```

```
create table Student (
   Name varchar2(28),
   Dept varchar2(20),
   Major varchar2(20),
   sex char(1));
```

```
insert into faculty values ('Smith', 'CS', 'Prof', 81000.00, 'F');
insert into faculty values ('Brown', 'CS', 'Assoc Prof', 75000.00, 'M');
insert into student  values ('Brown', 'CS', 'Info', 'F');
```

```
(SELECT Name, Dept FROM Faculty)
           UNION
(SELECT Name, Dept FROM Student);
NAME                DEPT
----------------- --------------------
Brown                 CS
Smith                 CS
```

```
(SELECT Name, Dept FROM Faculty)
           UNION ALL
(SELECT Name, Dept FROM Student);
NAME                        DEPT
------------------------ ------------------
Smith                        CS
Brown                        CS
Brown                        CS
```

Relation schemas:

TA (Name, stipend, course)

Student (Name, Dept, Major, sex)

Query: Find names of female TAs who are majoring in the department of Computer Science

```
(SELECT name
 FROM  TA)
            INTERSECT
(SELECT name
 FROM Student
 WHERE Dept="Computer Science"
            and sex = "F");
```

Find names of TAs **who are not** majoring in the department of Computer Science.

```
(SELECT name
     FROM TA)
                EXCEPT
 (SELECT Name
  FROM Student
 WHERE Dept = "Computer Science" );
```

A tuple in SQL is represented by a parenthesized list of scalar values; (Smith, ’CompSci’) or (Smith, Student.Dept)

If a tuple ***t*** has the same number of components as a table(relation) ***R***, then it makes sense to compare ***t*** and ***R***

t **IN** ***R*** -- this is true iff ***t*** ∈ ***R***

t <> **ANY** ***R*** -- *true if t is neither greater nor less than any tuple in* ***R***

Relation schemas: **Student** (Name, Dept, Major, sex),

TA (Name, stipend, course), GRADE(Name, course, gr)

Find students who got an A in the course where they are TAs

```
select t.Name
from TA t
where (t.name, t.course) in
                              (select Name, course
                               from grade
                                where gr = ‘A’);
```

We are conveniently ignoring time!

INSERT, DELETE, ALTER

The form of a insert statement:

insert into relation[list of attributes] values(list of values);

insert into relation *select statement*;

The form of a delete statement:

delete **from *relation* where <**condition**>;**

Delete every tuple in the relation satisfying the condition

The form of an update statement:

update *relation* set <new-value assignments**> where <**condition**>;**

Adding Columns

alter table relation add <column declaration>;

Removing Columns

Add & Remove with care!

alter table relation DROP <column name>;

select * from EMPL;

```
| Eid | Name     | title     | salary | emailsuffix      |
+-----+----------+-----------+--------+------------------+
|  33 | John     | SrProgr   | 120000 | coldmail.org     |
|  34 | Jenny    | SrProgr   | 110000 | coldmail.org     |
|  35 | Anne     | WebDev    |  90000 | gonemail.com     |
|  36 | Mary     | WebDev    |  85000 | comemail.com     |
|  37 | Freddy   | Progr     |  75000 | netmail.com      |
|  38 | Johnny   | Progr     |  80000 | netmail.com      |
|  39 | Art      | Progr     |  75000 | netmail.com      |
|  40 | Albert   | Progr     |  70000 | netmail.com      |
|  41 | Susan    | WebProgr  |  90000 | gonemail.com     |
|  42 | Paul     | WebProgr  |  85000 | gonemail.com     |
|  43 | Edward   | DBProgr   |  75000 | coldmail.org     |
|  44 | Kim      | WebDev    | 110000 | coldmail.org     |
|  45 | Roger    | DBA       | 150000 | comemail.com     |
|  46 | Danny    | NetAdmin  | 100000 | sizzlingmail.com |
|  47 | Mike     | Mkt Mgr   | 120000 | gonemail.com     |
|  48 | MaryAnne | Mkt Mg    |  90000 | speedymail.com   |
+-----+----------+-----------+--------+------------------+
```

Group by and **Having**

The Group by clause is to group the data by the value(s) of one (or more) column(s)
The predicate for the GROUP BY clause is HAVING

```
CREATE TABLE EMPL (
    Eid int unsigned not null auto_increment primary key,
    Name varchar(20),
    title varchar(30),
    salary int,
    emailsuffix varchar(60)
);

ALTER TABLE EMPL AUTO_INCREMENT = 100;
```

```
select title, count(*) AS HowMany
from EMPL
GROUP BY title
ORDER BY HowMany;
```

```
+----------+---------+
| title    | HowMany |
+----------+---------+
| DBA      |       1 |
| NetAdmin |       1 |
| Mkt Mgr  |       1 |
| Mkt Mg   |       1 |
| DBProgr  |       1 |
| SrProgr  |       2 |
| WebProgr |       2 |
| WebDev   |       3 |
| Progr    |       4 |
+----------+---------+
9 rows in set (0.00 sec)
```

```
select title, count(*) AS HowMany
from EMPL
GROUP BY title having count(title)>=2;
```

```
+----------+---------+
| title    | HowMany |
+----------+---------+
| Progr    |       4 |
| SrProgr  |       2 |
| WebDev   |       3 |
| WebProgr |       2 |
+----------+---------+
4 rows in set (0.00 sec)
```

```
select title, emailsuffix
from EMPL
GROUP BY title, emailsuffix;
```

```
+----------+------------------+
| title    | emailsuffix      |
+----------+------------------+
| DBA      | comemail.com     |
| DBProgr  | coldmail.org     |
| Mkt Mg   | speedymail.com   |
| Mkt Mgr  | gonemail.com     |
| NetAdmin | sizzlingmail.com |
| Progr    | netmail.com      |
| SrProgr  | coldmail.org     |
| WebDev   | coldmail.org     |
| WebDev   | comemail.com     |
| WebDev   | gonemail.com     |
| WebProgr | gonemail.com     |
+----------+------------------+
11 rows in set (0.00 sec)
```

```
select title, emailsuffix
from EMPL
GROUP BY title, emailsuffix
having count(title)>2;
```

```
+-------+-------------+
| title | emailsuffix |
+-------+-------------+
| Progr | netmail.com |
+-------+-------------+
 select title, emailsuffix
 from EMPL
 GROUP BY title, emailsuffix
 having count(title)>=2
 ORDER BY title;
```

```
+----------+--------------+
| title    | emailsuffix  |
+----------+--------------+
| Progr    | netmail.com  |
| SrProgr  | coldmail.org |
| WebProgr | gonemail.com |
+----------+--------------+
3 rows in set (0.01 sec)
```

```
select title, salary
from EMPL
where emailsuffix like '%org'
GROUP BY title, emailsuffix
having count(title)>=2
ORDER BY salary;
```

```
+----------+--------+
| title    | salary |
+----------+--------+
| Progr    |  75000 |
| WebProgr |  90000 |
+----------+--------+
2 rows in set (0.00 sec)
```

Merge rows

Using the select statement to merge multiple rows into 1 row:
 MySQL: the **group_concat** notation".

```
mysql> select C, group_concat(B) as Bs
    -> from R
    -> group by C;
+-------+--------+
| C     | Bs     |
+-------+--------+
|    12 | 10,11  |
|    14 | 9      |
|    17 | 8      |
+-------+--------+
3 rows in set (0.03 sec)
```

```
mysql> select * from R;
+----+------+------+
| A  | B    | C    |
+----+------+------+
| a1 |   10 |   12 |
| a2 |   11 |   12 |
| a3 |    9 |   14 |
| a4 |    8 |   17 |
+----+------+------+
```

Merge rows

Using the select statement to merge multiple rows into 1 row:
 MySQL: the **group_concat** notation".

```
mysql> select C, group_concat(distinct B separator ‘;’ ) as Bs
    -> from R
    -> group by C;
+-------+--------+
| C     | Bs     |
+-------+--------+
|    12 | 10;11  |
|    14 | 9      |
|    17 | 8      |
+-------+--------+
3 rows in set (0.03 sec)
```

two **10** are not repeated

```
mysql> select * from R;
+----+------+------+
| A  | B    | C    |
+----+------+------+
| a1 |   10 |   12 |
| a2 |   11 |   12 |
| a3 |    9 |   14 |
| a4 |    8 |   17 |
| a5 |   10 |   12 |
+----+------+------+
```

Merge rows: Oracle

```
SELECT C,
 listagg(B, ', ') WITHIN GROUP (ORDER BY B) AS Bs
FROM R
GROUP BY C;
```

```
select * from R;
+----+------+------+
| A  | B    | C    |
+----+------+------+
| a1 |   10 |   12 |
| a2 |   11 |   12 |
| a3 |    9 |   14 |
| a4 |    8 |   17 |
+----+------+------+
```

```
C       Bs
12      10, 11
14      9
17      8
3 rows returned in 0.07 seconds
```

Update part of text in a column

Handy to update part of an existing text column in a table!

```
select message  from account_email where message like '%confsys%';
11 rows in set (0.001 sec)

update account_email
 set message =replace (message ,'confsys.encs', 'ideas.encs');
Query OK, 11 rows affected (0.010 sec)

select message  from account_email where message like '%confsys%';
Empty set (0.001 sec)
```

Revert:

```
update account_email
    ->  set message =replace (message ,'ideas.encs','confsys.encs');
Query OK, 11 rows affected (0.009 sec)
```

Functions

Most database systems have a multitude of functions:

- Comparison Functions and Operators
- Logical Operators
- Control Flow Functions
- String Functions
- Mathematical Functions
- Date and Time Functions
- Encryption and Compression Functions
- Bit Functions
- Full-Text Search Functions
- Cast functions and Operators
- Information Functions
- XML Functions

Regular expression

REGEXP is used to give a pattern scheme for a string comparison of the pattern with a string using the syntax:

```
expr REGEXP pat
```

If there is a match the REGEXP function returns 1, else 0.

If either expression or pattern is NULL, the function returns NULL.

Some meta-characters: ^, * , . , [ ] , ( ) , {m,n}

^ Match the beginning of a string.
$ Match the end of a string.
. Match any character

Suggestion: Look up manual/tutorials and try examples.

Date format for MySQL is YYYY-MM-DD - the SQL2 default

To set Oracle’s default date format to YYYY-MM-DD

Internally Oracle stores both date and time as a single value

```
 $conn = OCILogon($my_Ora_id,$My-Ora_PW,$My_Ora_db)
 //Set Oracle’s date format to YYYY-MM-DD
 $stmt = OCIParse($conn,"ALTER SESSION SET
                    NLS_DATE_FORMAT='YYYY-MM-DD'");
OCIExecute($stmt,OCI_DEFAULT);
```

```
create table bdate( Name char(25), bday date);
insert into bdate values('Jane', '20-Jan-83');
select * from bdate;
NAME                      BDAY
-------------------------  ---------
Jane                      20-JAN-83
```

```
MariaDB>insert into bdate values('Jane', '1983-01-20');
```

```
ALTER SESSION SET NLS_DATE_FORMAT='YYYY-MM-DD';
SQL> select * from bdate;
NAME                        BDAY
-------------------------   ----------
Jane                        1983-01-20

SELECT Name, TO_CHAR(bday, 'YYYY/MM/DD') AS Birthday
FROM bdate;
NAME                      BIRTHDAY
------------------------- ----------
Jane                      1983/01/20
```

In Oracle:
The functions TO_CHAR or TO_DATE return part of the date/time.
TRUNC will return the first day of the period. ROUND will round up at mid year/mid month (July 1 or 16th day)

```
CREATE TABLE supplies
    (supname  char(14),
    part#     number(4),
    price     number(7,2));
```

```
CREATE TABLE project
    (proj# number(4),
    projname  char(14),
    projmgr   number(4));
```

```
CREATE TABLE usedin
    (proj#    number(4),
    part#     number(4),
    qty       number(3));
```

```
CREATE TABLE empls
    (emp#     number(4),
    empname   char(14),
    address   char(14));
```

```
SQL> select * from empls;

      EMP# EMPNAME         ADDRESS
---------- --------------- ------------
       120 Hardrock        Outremont
       123 Eliza           NDG
       124 John            Laval
       127 Jim             Montreal
       129 Sun             Brossard
       131 Moon            Beaconsfield
       135 Dr. Dolittle    Laval
       141 Knowit          Montreal
       142 Softee          NDG
       143 Dr. Knowall     Montreal
```

```
SQL> select * from assigned_to;

     PROJ#       EMP#      HOURS
---------- ---------- ----------
       353        135         20
       753        135         20
       353        123          6
       353        124         40
       451        141         40
       753        127         40
       353        129          4
       451        131         10
       321        120         40
       326        142         40
       326        129         36
       451        135          1
```

```
SQL> select * from project;

     PROJ# PROJNAME           PROJMGR
---------- --------------- ----------
       353 database               135
       451 database               141
       321 software               120
       326 hardware               142
       753 database               135
```

```
     EMP# EMPNAME         ADDRESS
--------- --------------- ----------
      135 Dr. Dolittle    Laval
```

USEDIN

COMP321	1	5
COMP321	2	2
COMP321	3	3
COMP326	4	1
COMP326	5	3
COMP326	6	4
COMP353	1	5
COMP353	8	1
COMP451	9	2
COMP451	7	3
COMP753	1	4
COMP753	2	3
COMP753	3	6
COMP753	4	4

SUPPLIER

PROVIBEC	Quebec
SUPORIO	Toronto
MANIPART	Winnipeg
SUPBEC	Laval
NDG-SUPPLY	NDG

SUPPLIES

```
SUPNAME              PART#      PRICE
-------------- ---------- ----------
PROVIBEC                1      710.2
PROVIBEC                2      815.3
PROVIBEC                3        325
PROVIBEC                4     795.99
SUPORIO                 1     695.99
SUPBEC                  2     799.98
NDG-SUPPLY              1     699.99
NDG-SUPPLY              2     799.99
NDG-SUPPLY              3     324.99
NDG-SUPPLY              4     795.98
NDG-SUPPLY              7        754
SUPBEC                  1     699.98
MANIBEC                 1     727.99
MDG-SUPPLY              1     699.99
MDG-SUPPLY              2     799.99
MDG-SUPPLY              3     324.99
MDG-SUPPLY              4     795.98
```

PROJECT(***Project#***, *Project_Name*,*Chief_Architect*)
EMPLOYEE (***Emp#***, *EmpName*)
ASSIGNED_TO (***Project#***, ***Emp#***)

◆Get details of employees working on **all** Database projects.

$$
\begin{aligned}
\{s \mid\ & s \in \text{employee} \\
& \wedge\ \forall t(t \in \text{project} \wedge t[\mathit{Project\_Name}] = \text{'Database'} \\
& \rightarrow \exists u(u \in \text{assigned\_to} \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
& \wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}]))\}
\end{aligned}
$$

replacing $f \rightarrow g$ by its equivalent form $\neg f \vee g$:

$$
\begin{aligned}
\{s \mid\ & s \in \text{employee} \\
& \wedge\ \forall t(t \notin \text{project} \vee t[\mathit{Project\_Name}] \neq \text{'Database'} \\
& \vee\ \exists u(u \in \text{assigned\_to} \wedge\ u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
& \wedge\ s[\mathit{Emp\#}] = u[\mathit{Emp\#}]))\}
\end{aligned}
$$

Using $\forall x(P(x)) \equiv \neg(\exists x)(\neg P(x))$ **Assertion of Universality**

$$\{s \mid s \in \text{employee}$$
$$\wedge \neg(\exists t)(\neg (t \notin \text{project} \vee t[\textit{Project\_Name}] \neq \text{'Database'}$$
$$\vee \exists u(u \in \text{assigned\_to} \wedge u[\textit{Project\#}] = t[\textit{Project\#}]$$
$$\wedge s[\textit{Emp\#}] = u[\textit{Emp\#}])))\}$$

$$\{s \mid s \in \text{employee}$$
$$\wedge \neg(\exists t)\ (t \in \text{project} \wedge t[\textit{Project\_Name}] = \text{'Database'}$$
$$\neg(^{\vee} \exists u(u \in \text{assigned\_to} \wedge u[\textit{Project\#}] = t[\textit{Project\#}]$$
$$\wedge s[\textit{Emp\#}] = u[\textit{Emp\#}])\ ))\}$$

$$\{s \mid s \in \text{employee}$$
$$\wedge \neg(\exists t)\ (t \in \text{project} \wedge t[\textit{Project\_Name}] = \text{'Database'}$$
$$\wedge \neg \exists u(u \in \text{assigned\_to} \wedge u[\textit{Project\#}] = t[\textit{Project\#}]$$
$$\wedge s[\textit{Emp\#}] = u[\textit{Emp\#}])\ )\}$$

Get details of employees working on all Database projects.

$$\begin{aligned}
&\{s \mid s \in \text{employee} \\
&\qquad \land \neg(\exists t)\,(t \in \text{project} \land t[\mathit{Project\_Name}] = \text{'Database'} \\
&\qquad \land \neg\, \exists u(u \in \text{assigned\_to} \land\ u[\mathit{Project\#}] = t[\mathit{Project\#}] \\
&\qquad \land s[\mathit{Emp\#}] = u[\mathit{Emp\#}])\ )\}
\end{aligned}$$

```
select *
from employees s
where not exists ( select *
                   from project t
                    where t.Project_Name = 'Database'  and
                    not exists(select *
                               from assigned_to u
                                where u.Project# = t.Project#  and
                                 s.Emp# = u.Emp#))
```

ALTERNATE SCHEME:

Using set operation for answering “ALL” type queries

```
select *
from empls s
where not exists( (select Proj#
                        from project t
                         where t.ProjName = 'database')
            minus
                  (select u.Proj#
                     from assigned_to u, project t1
                      where u.Proj# = t1.Proj#
                          and s.EMP# = u.Emp#
                           and t1.ProjName = 'database'));
```

Set of all projects with name = ‘database’

Set of projects with name=‘database’ assigned to employee s

USE OF VIEW

A view is a materialized(virtual) table that can be used in any SQL query

Create a view (project numbers) of the projects managed by an employee with the name 'Dr. Dolittle'

Optional renaming of attributes used in view definitions

```
create view do_project[(projnumber)] as
select p.proj#
from project p
where p.projmgr = (select e.emp#
                               from empls e
                              where empname='Dr. Dolittle')
```

Using VIEW

Find suppliers who can supply all parts used in a project managed by Dr. Dolittle, and the corresponding project number(s)

$$
\begin{aligned}
\{S \mid s \in \text{supplies} \wedge d \in \text{do\_project} \\
&\wedge S[\mathit{Supname}] = s\,[\mathit{Supname}] \\
&\wedge\ S[\mathit{Project\#}] = d\,[\mathit{Projectnumb}] \\
&\wedge \neg(\exists u)\,(u \in \text{used\_in} \wedge u[\mathit{Project\#}] = d[\mathit{Projectnumb}] \\
&\wedge \neg\ \exists t(t \in \text{supplies} \wedge\ u[\mathit{Part\#}] = t[\mathit{Part\#}] \\
&\wedge t[\mathit{Supname}] = s[\mathit{Supname}]\ ))\}
\end{aligned}
$$

A view can appear where a relation name is allowed

Find suppliers who can supply all parts used in a project managed by Dr. Dolittle

```
select unique s.supname, d.proj#
from supplies s, do_project d
where exists
    (select *
     from supplies s1
     where s1.supname=s.supname
     and not exists
       (select *
        from usedin u
        where u.proj# = d.proj#
         and not exists
                (select *
                from supplies s2
                where s2.supname = s1.supname
                            and s2.part#=u.part#)))
```

Is this predicate required??

There exists a supplier ***s*** and project **d** such that there are no parts used in **d** that is not supplied by this supplier ***s***

```
SUPNAME         PROJ#
--------------- ----
MDG-SUPPLY      753
NDG-SUPPLY      753
PROVIBEC        753
```

A view can appear where a relation name is allowed

Find suppliers who can supply all parts used in a project managed by Dr. Dolittle

```
select unique s.supname, d.proj#
from supplies s, do_project d
where not exists
  (select *
    from usedin u
    where u.proj# = d.proj#
    and not exists
      (select *
        from supplies s2
        where s2.supname = s.supname
        and s2.part#=u.part#))
```

There exists a supplier ***s*** and project **d** such that there are no parts used in **d** that is not supplied by this supplier ***s***

```
SUPNAME              PROJ#
---------------      ----
MDG-SUPPLY           753
NDG-SUPPLY           753
PROVIBEC             753
```

An Alternative

Using the view do_project and set difference operations to write the SQL query for:

Find suppliers who can supply all parts used in a project managed by Dr. Dolittle

```
SUPNAME          PROJ#
---------------- ----
MDG-SUPPLY       753
NDG-SUPPLY       753
PROVIBEC         753
```

```
select unique s.supname, d.proj#
from supplies s, do_project d
where not exists
   (select u.part#
     from usedin u
      where u.proj# = d.proj#
       minus (select s2.part#
                    from supplies s2
                     where s2.supname = s.supname))
```

Parts used in one of Dolittle's project

Parts supplied by s

Find suppliers who can supply all parts used in a project managed by Dr. Dolittle

```
select unique s.supname, d.proj#
from supplies s, do_project d
where not exists
   (select s2.part#
               from supplies s2
                where s2.supname = s.supname
         minus
         (select u.part#
            from usedin u
             where u.proj# = d.proj# )))
```

Parts supplied by s

Parts used in one of Dolittle’s project

What is wrong with this query??

If a supplier supplies all parts used in the project but also other parts than this supplier would not be included.

How to update a view?

Translate modification of the view to the corresponding modification on the base tables used in the view definition –be able to identify the base relation(s) and attribute(s)

Should we allow updates on views?

Yes, however it depends - some problems can arise

Some simple views can be updated

Known as **updatable views (easy if primary keys are part of view)**

Many views cannot be updated

This is due to the so called **view-update anomaly**

insert into do_project **values(**‘Proj1’**);** **⇒(Proj1, null, null)**

Note: null should be allowed for the base attributes

***Would the insertion cause the insertion of* (Proj1, null, emp# of Dr. Do..)?**

SQL provides a formal definition of when modifications to a view are permitted

- it is permitted if the view is defined by selecting some attributes from one relation **R**, which could be an “updatable” view itself

- the view definition uses **SELECT** (but not **SELECT DISTINCT**)

- the **WHERE** clause does not involve **R** in a sub query

- the list in the **SELECT** clause includes “enough” attributes that for every tuple inserted into the view, the tuple inserted into the base relation will “yield” the inserted tuple of the view

- the **NOT NULL** constraints on the base relation will not be violated

SQL allows user defined data types - **domains**

We can define a domain as follows:

create domain <name> **as** <type description>**default** ***value*****;**

To create a domain with default value:

```
        create domain Projnumbers as number(4) default 9999;
```

To change the default for a domain:

```
        alter domain Projnumbers set default 0;
```

To delete a domain definition:

```
        drop domain Projnumbers;
```

Arithmetic operations on NULLs

Result of an arithmetic operator, when at least one of the operands has a value of NULL, is NULL

if x have the value NULL, then x+3 is also NULL

However, NULL is not a constant

NULL + 3 is ***illegal***

Some basic arithmetic rules are not applicable.

Suppose x is a numeric value

$x * 0 = 0$, but if x is NULL then $x * 0$ is NULL

$x - x = 0$, but if x is NULL then $x - x$ is NULL

In 3-Valued Logic we may assume that:

TRUE = 1, **FALSE** = 0 , **UNKNOWN** = 1/2

x **AND** *y* = min (*x*, *y*) , *x* **OR** *y* = max (*x*, *y*) , **NOT** *x* = 1-*x*

X	***Y***	***X* AND *Y***	***X* OR *Y***	**NOT *X***
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

Null value and logical operations

Two value logic: *x* **OR (NOT** *x***)** = 0 OR 1| 1 OR 0 = 1 = **TRUE**

For 3-valued logic:

x **OR (NOT** *x***)** = **max**(1/2,(1**-**1/2)) = 1/2 = **UNKNOWN** ≠ **(TRUE)**

Note: We can't treat NULL as a constant :

grade (Name, course, gr)

Consider query:

```
select *
from grade
WHERE gr <= “c” or gr > “c” ;
```

Here the result is expected to be the **grade** relation.

If null values are allowed for gr, then the above query returns only tuples of **grade** where the value of gr is not NULL .

In SQL2, there are other forms for expressing × and ⋈

Cartesian Product of Employee and Position

Employee (*Empl_No*, *Name*, *Skill*, *Pay_rate*)   Position (*Posting_No*, *Skill*)

select * from Employee, Position;

select * from Employee CROSS JOIN position;

Theta join of Employee, Position

select * from Employee **JOIN** Position **ON**
Position. *Skill* = Employee. *Skill;*

```
select *
 from Employees
 JOIN Position
 USING (skill);
```

select distinct * from Employee **JOIN** Position **ON**
Position. *Skill* = Employee. *Skill;*

To remove duplicates

Natural join of Employee, Position

select * from Employee **NATURAL JOIN** Position

OUTER JOIN -- computes the join relations preserving **dangling** tuples by padding them with **NULLs**

A tuple in R is dangling if it doesn't join with any tuple in S; similarly a tuple in S is dangling if it doesn't join with any tuple in R

FULL OUTER JOIN: It pads dangling tuples of ***R*** and ***S*** preserving them

LEFT OUTER JOIN: It pads dangling tuples of ***R*** only; tuples of ***R*** preserved

RIGHT OUTER JOIN: It pads dangling tuples of ***S*** only; tuples of ***S*** preserved

R:

A	B
1	2
2	3

S:

B	C
2	5
2	6
7	8

R FULL OUTER JOIN S:

A	B	C
1	2	5
1	2	6
2	3	⊥
⊥	7	8

R LEFT OUTER JOIN S:

A	B	C
1	2	5
1	2	6
2	3	⊥

R RIGHT OUTER JOIN S:

A	B	C
1	2	5
1	2	6
⊥	7	8

In SQL

R [NATURAL] [LEFT | RIGHT | FULL] OUTER JOIN S [ON ...]

```
SQL> select * from part;
     PART# DESCR
---------- ----------
         1 part1
         2 part2
         3 part3
         4 part4
         5 part5
         6 part6
         7 part7
         8 part8
         9 part9
```

```
SQL> select * from supplies;
SUPNAME             PART#      PRICE
-------------- ---------- ----------
PROVIBEC                1      710.2
PROVIBEC                2      815.3
PROVIBEC                3        325
PROVIBEC                4     795.99
SUPORIO                 1     695.99
SUPBEC                  2     799.98
NDG-SUPPLY              1     699.99
NDG-SUPPLY              2     799.99
NDG-SUPPLY              3     324.99
NDG-SUPPLY              4     795.98
NDG-SUPPLY              7        754
SUPBEC                  1     699.98
MANIBEC                 1     727.99
MDG-SUPPLY              1     699.99
MDG-SUPPLY              2     799.99
MDG-SUPPLY              3     324.99
MDG-SUPPLY              4     795.98
```

```
select *
from part p left outer join supplies s
on (p.part# = s.part#);

     PART# DESCR      SUPNAME              PART#      PRICE
---------- ---------- --------------- ---------- ----------
         1 part1      PROVIBEC                 1      710.2
         2 part2      PROVIBEC                 2      815.3
         3 part3      PROVIBEC                 3        325
         4 part4      PROVIBEC                 4     795.99
         1 part1      SUPORIO                  1     695.99
         2 part2      SUPBEC                   2     799.98
         1 part1      NDG-SUPPLY               1     699.99
         2 part2      NDG-SUPPLY               2     799.99
         3 part3      NDG-SUPPLY               3     324.99
         4 part4      NDG-SUPPLY               4     795.98
         7 part7      NDG-SUPPLY               7        754
         1 part1      SUPBEC                   1     699.98
         1 part1      MANIBEC                  1     727.99
         1 part1      MDG-SUPPLY               1     699.99
         2 part2      MDG-SUPPLY               2     799.99
         3 part3      MDG-SUPPLY               3     324.99
         4 part4      MDG-SUPPLY               4     795.98
         5 part5
         8 part8
         6 part6
         9 part9

21 rows selected.
```

```
select *
from part p right  outer join supplies s
on (p.part# = s.part#);

     PART# DESCR      SUPNAME             PART#      PRICE
---------- ---------- -------------- ---------- ----------
         1 part1      MDG-SUPPLY              1     699.99
         1 part1      MANIBEC                 1     727.99
         1 part1      SUPBEC                  1     699.98
         1 part1      NDG-SUPPLY              1     699.99
         1 part1      SUPORIO                 1     695.99
         1 part1      PROVIBEC                1      710.2
         2 part2      MDG-SUPPLY              2     799.99
         2 part2      NDG-SUPPLY              2     799.99
         2 part2      SUPBEC                  2     799.98
         2 part2      PROVIBEC                2      815.3
         3 part3      MDG-SUPPLY              3     324.99
         3 part3      NDG-SUPPLY              3     324.99
         3 part3      PROVIBEC                3        325
         4 part4      MDG-SUPPLY              4     795.98
         4 part4      NDG-SUPPLY              4     795.98
         4 part4      PROVIBEC                4     795.99
         7 part7      NDG-SUPPLY              7        754

17 rows selected.
```

```
SQL> select * from part p full outer join supplies s on (p.part# =
s.part#);
     PART# DESCR      SUPNAME             PART#      PRICE
---------- ---------- -------------- ---------- ----------
         1 part1      PROVIBEC                1      710.2
         2 part2      PROVIBEC                2      815.3
         3 part3      PROVIBEC                3        325
         4 part4      PROVIBEC                4     795.99
         1 part1      SUPORIO                 1     695.99
         2 part2      SUPBEC                  2     799.98
         1 part1      NDG-SUPPLY              1     699.99
         2 part2      NDG-SUPPLY              2     799.99
         3 part3      NDG-SUPPLY              3     324.99
         4 part4      NDG-SUPPLY              4     795.98
         7 part7      NDG-SUPPLY              7        754
         1 part1      SUPBEC                  1     699.98
         1 part1      MANIBEC                 1     727.99
         1 part1      MDG-SUPPLY              1     699.99
         2 part2      MDG-SUPPLY              2     799.99
         3 part3      MDG-SUPPLY              3     324.99
         4 part4      MDG-SUPPLY              4     795.98
         5 part5
         8 part8
         6 part6
         9 part9
```

Joins in SQL

```
mysql> select * from R;  mysql> select * from S;
+----+------+------+          +------+------+----+
| A  | B    | C    |          | B    | C    | D  |
+----+------+------+          +------+------+----+
| a1 |   10 |   12 |          |   10 |   12 | d1 |
| a2 |   11 |   12 |          |   11 |   12 | d2 |
| a3 |    9 |   14 |          |    6 |   14 | d3 |
| a4 |    8 |   17 |          |    9 |   12 | d4 |
+----+------+------+          +------+------+----+
```

In MySQL(up to version 5.7 at least), JOIN, CROSS JOIN, and INNER JOIN are syntactic equivalents: i.e., they can be used interchangeably.

In standard SQL, they are not equivalent.
INNER JOIN is used with an ON clause,

```
select *  from Employee JOIN Position ON
                        Position. Skill = Employee. Skill;
```

```
select R.a, T .e
from R inner join S on R.b = S.b
          inner join T on S.c = T.c
```

```
MariaDB [test]> select *
        from R join S
         on R.B=S.B;
+---+---+---+---+---+---+
| A | B | C | B | C | D |
+---+---+---+---+---+---+
| a1| 10| 12| 10| 12| D1|
| a2| 11| 12| 11| 12| D2|
| a3|  9| 15|  9| 12| D4|
+---+---+---+---+---+---+
3 rows in set (0.028 sec)
```

CROSS JOIN is used as follows:

```
select *
from R cross join S;
```

ALL THE FOLLOWING ARE EQUIVALENT IN MySQL

Note: result schema R || S

```
select *
from R JOIN S;

select *
from R, S;

select *
from R CROSS JOIN S;

select *
from R INNER JOIN S;
```

4*4 rows in result

```
+----+----+----+-----+-----+----+
| A  | B  | C  | B   | C   | D  |
+----+----+----+-----+-----+----+
| a1 | 10 | 12 |  10 |  12 | d1 |
| a2 | 11 | 12 |  10 |  12 | d1 |
| a3 |  9 | 14 |  10 |  12 | d1 |
| a4 |  8 | 17 |  10 |  12 | d1 |
| a1 | 10 | 12 |  11 |  12 | d2 |
| a2 | 11 | 12 |  11 |  12 | d2 |
| a3 |  9 | 14 |  11 |  12 | d2 |
| a4 |  8 | 17 |  11 |  12 | d2 |
| a1 | 10 | 12 |   6 |  14 | d3 |
| a2 | 11 | 12 |   6 |  14 | d3 |
| a3 |  9 | 14 |   6 |  14 | d3 |
| a4 |  8 | 17 |   6 |  14 | d3 |
| a1 | 10 | 12 |   9 |  12 | d4 |
| a2 | 11 | 12 |   9 |  12 | d4 |
| a3 |  9 | 14 |   9 |  12 | d4 |
| a4 |  8 | 17 |   9 |  12 | d4 |
+----+----+----+-----+-----+----+
```

E**quijoin:** join predicate containing an equality operator.
- combines rows that have same values for the specified columns.

If two tables in a join query have no join predicate the DBMS returns a **Cartesian product**.

Outer Join

An outer join extends the result of a simple join.
An **outer join** returns all rows that satisfy the join condition and those rows from one table for which no rows from the other satisfy the join condition.
Such rows are not returned by a simple join.

Joins: Equi-Joins

```
select * from R JOIN S on R.B=S.B;
| A  | B    | C    | B    | C    | D  |
+----+------+------+------+------+----+
| a1 |   10 |   12 |   10 |   12 | d1 |
| a2 |   11 |   12 |   11 |   12 | d2 |
| a3 |    9 |   14 |    9 |   12 | d4 |

select * from R JOIN S on R.B=S.B and R.C=S.C;
| A  | B    | C    | B    | C    | D  |
+----+------+------+------+------+----+
| a1 |   10 |   12 |   10 |   12 | d1 |
| a2 |   11 |   12 |   11 |   12 | d2 |
select * from R left outer join S on R.B=S.B;
| A   | B      | C      | B      | C      | D      |
+----+-------+-------+-------+-------+-------+
| a1 |    10 |    12 |    10 |    12 | d1    |
| a2 |    11 |    12 |    11 |    12 | d2    |
| a3 |     9 |    14 |     9 |    12 | d4    |
| a4 |     8 |    17 | NULL  | NULL  | NULL  |
```

```
select * from R left outer join S
on R.B=S.B and R.C=S.C;
| A   | B      | C      | B      | C      | D      |
+----+-------+-------+-------+-------+-------+
| a1 |    10 |    12 |    10 |    12 | d1    |
| a2 |    11 |    12 |    11 |    12 | d2    |
| a3 |     9 |    14 | NULL  | NULL  | NULL  |
| a4 |     8 |    17 | NULL  | NULL  | NULL  |
select *
from R right outer join S
on R.B=S.B and R.C=S.C;
```

```
+------+------+------+------+------+----+
| A    | B    | C    | B    | C    | D  |
+------+------+------+------+------+----+
| a1   |   10 |   12 |   10 |   12 | d1 |
| a2   |   11 |   12 |   11 |   12 | d2 |
| NULL | NULL | NULL |    6 |   14 | d3 |
| NULL | NULL | NULL |    9 |   12 | d4 |
+------+------+------+------+------+----+
```

FULL outer join does not exist in MySQL; Simulated by:

```
select *
from R left outer join S on R.B=S.B
UNION
select *
from R right outer join S on R.B=S.B;
+------+------+------+------+------+------+
| A    | B    | C    | B    | C    | D    |
+------+------+------+------+------+------+
| a1   |   10 |   12 |   10 |   12 | d1   |
| a2   |   11 |   12 |   11 |   12 | d2   |
| a3   |    9 |   14 |    9 |   12 | d4   |
| a4   |    8 |   17 | NULL | NULL | NULL |
| NULL | NULL | NULL |    6 |   14 | d3   |
+------+------+------+------+------+------+
select *
from R left outer join S
on R.B=S.B and R.C=S.C
UNION
select *
from R right outer join S
on R.B=S.B and R.C=S.C;
```

FULL outer join does not exist in MySQL;

Simulated by:

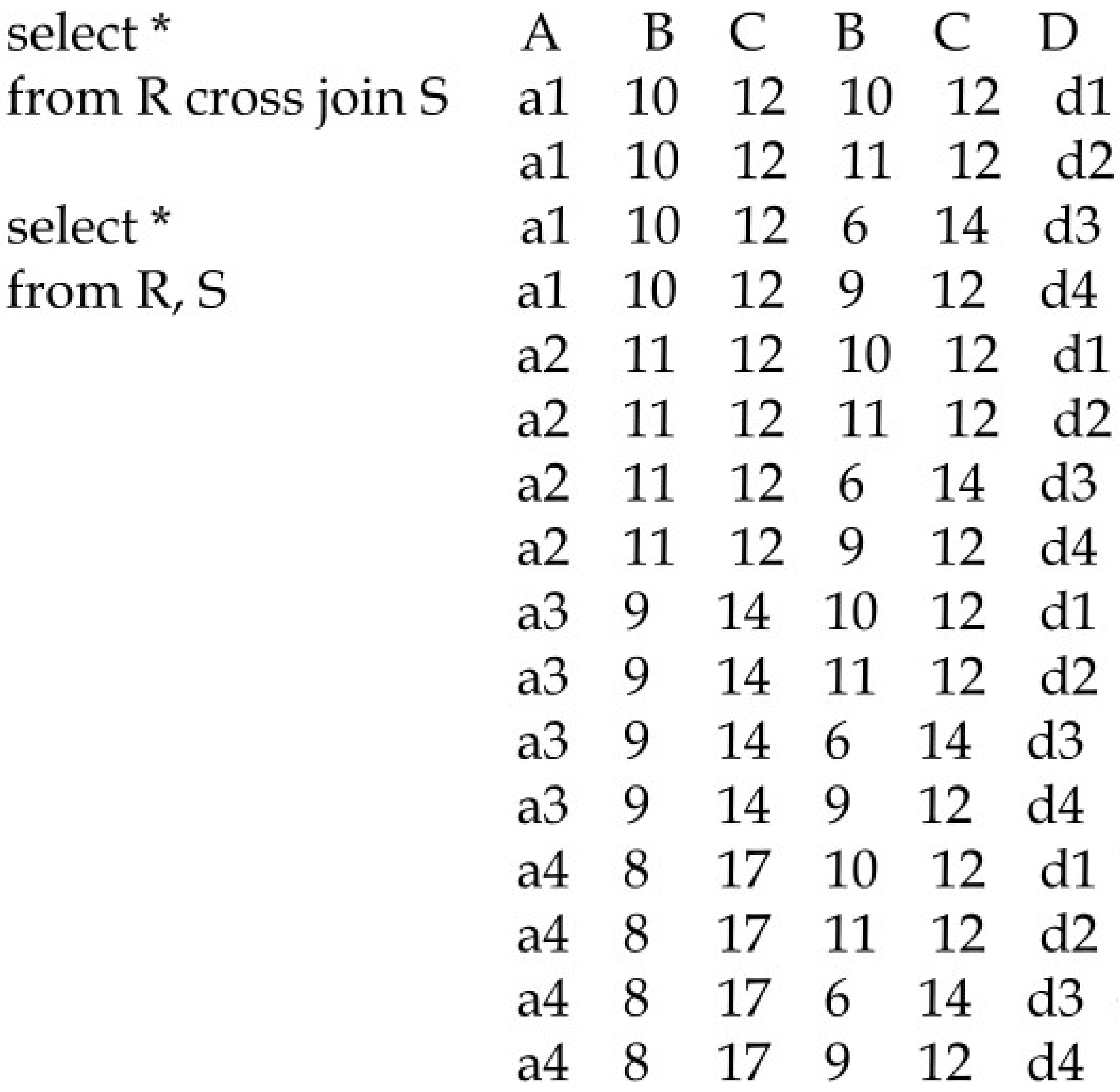
```
+-------+-------+-------+-------+-------+-------+
| A     | B     | C     | B     | C     | D     |
+-------+-------+-------+-------+-------+-------+
| a1    |    10 |    12 |    10 |    12 | d1    |
| a2    |    11 |    12 |    11 |    12 | d2    |
| a3    |     9 |    14 | NULL  | NULL  | NULL  |
| a4    |     8 |    17 | NULL  | NULL  | NULL  |
| NULL  | NULL  | NULL  |     6 |    14 | d3    |
| NULL  | NULL  | NULL  |     9 |    12 | d4    |
+-------+-------+-------+-------+-------+-------+
```

Full outer Join: Oracle

```
select *
from R r
full outer join
S s
on r.B=s.B and r.C=s.C
```

```
A     B     C     B     C     D
a1    10    12    10    12    d1
a2    11    12    11    12    d2
-     -     -     6     14    d3
-     -     -     9     12    d4
a3    9     14    -     -     -
a4    8     17    -     -     -
6 rows returned in 0.01 seconds
```

The Full outer join simulation, as in MySQL causes problems with column headings: the ones used would be from the system catalog!



```
select *
from R cross join S

select *
from R, S
```

A	B	C	B	C	D
a1	10	12	10	12	d1
a1	10	12	11	12	d2
a1	10	12	6	14	d3
a1	10	12	9	12	d4
a2	11	12	10	12	d1
a2	11	12	11	12	d2
a2	11	12	6	14	d3
a2	11	12	9	12	d4
a3	9	14	10	12	d1
a3	9	14	11	12	d2
a3	9	14	6	14	d3
a3	9	14	9	12	d4
a4	8	17	10	12	d1
a4	8	17	11	12	d2
a4	8	17	6	14	d3
a4	8	17	9	12	d4

Oracle: Cross Join

- explicit and implicit

However, Oracle some releases get chocked up by:

```
select *
from R JOIN S;

select *
from R  inner join  S
```

Natural Join

```
           R
+----+------+------+
| A  | B    | C    |
+----+------+------+
| a1 |   10 |   12 |
| a2 |   11 |   12 |
| a3 |    9 |   14 |
| a4 |    8 |   17 |
+----+------+------+
```

```
             S
+-------+------+----+
| B     | C    | D  |
+-------+------+----+
|    10 |   12 | d1 |
|    11 |   12 | d2 |
|     6 |   14 | d3 |
|     9 |   12 | d4 |
+-------+------+----+
```

```
select *
from R  natural  join  S
```

```
B       C       A       D
10      12      a1      d1
11      12      a2      d2
2 rows returned in 0.03 seconds
```

Self Join

```
              R                                   S
+----+------+------+      +------+------+----+
| A  | B    | C    |      | B    | C    | D  |
+----+------+------+      +------+------+----+
| a1 |   10 |   12 |      |   10 |   12 | d1 |
| a2 |   11 |   12 |      |   11 |   12 | d2 |
| a3 |    9 |   14 |      |    6 |   14 | d3 |
| a4 |    8 |   17 |      |    9 |   12 | d4 |
+----+------+------+      +------+------+----+
```

To avoid duplicate

```
MariaDB [test]> select r1.A, r2.A, r1.B, r1.C
    -> from R r1
    -> inner join R r2 on r1.C=r2.C
    -> where r1.A < r2.A
    -> order by r1.A, r2.A
```

```
+----+----+------+------+
| A  | A  | B    | C    |
+----+----+------+------+
| a1 | a2 |   10 |   12 |
+----+----+------+------+
```

Join Operation Execution

Hash joins

- In a *hash* join, a DBMS does a full-scan of one of the tables in the join operation to build a main-memory hash table. Then it searches for a matching value in the (hash table) for the other table.

Hash joins need more main memory but it could execute faster for certain types of join, the hash join will execute faster than a nested loop join.

Join Operation Execution

Nested loops join

- The *nested loops* join is one of the original join pans and it is the most common method. In a *nested loops* join, we have two tables: one is the left operand table and the other the right hand table.

The an index for the attribute of left table is accessed to get the row IDs of the rows with the attribute value.

The matching rows of the second table are then probed in a nested loop and matching rows of the two tables are joined using an index range scan.

Join Operation Execution

Some queries will perform faster with NESTED LOOPS joins, some with HASH joins, while others favor sort-merge joins.

It is difficult to predict what join technique will be fastest *a priori*, so many tuning a database is to test the often use joins and record the statistics to guide the productions operations

Constraints

Primary keys declarations

Foreign key constraints is a referential integrity constraints

If a supplier supplies part# 4, then we must have **part# 4** in the table for part

Primary key constraint is declared within the DDL SQL command **CREATE TABLE** using the keywords **PRIMARY KEY** or **UNIQUE**

Many DBMSs treat them as synonyms: A table may have one primary key but any number of "unique" declarations

unique

```
CREATE TABLE project
 (proj# number(4) primary key,
    projname  char(14),
    projmgr   number(4));
```

```
CREATE TABLE assigned_to
(proj#    number(4),
  emp#       number(4),
  hours      number(3)
  primary key(proj#,emp#));
```

```
SQL>  create table x
 (a number (4) primary key);
Table created.
SQL> create table y
  (s number(4),
   b number (4) references x(a));
Table created.

SQL> insert into y values(1, 2);
insert into y values(1, 2)
*
ERROR at line 1:
SQL> insert into x values (2);
1 row created.
SQL> insert into y values(1, 2);
1 row created.
```

NOTE: References must be a primary key or an attribute with an unique attribute

```
SQL> create table x(
  2  a number (4) primary key,
  3  b number(4));
Table created.
SQL> create table y(
  2  c number (4) primary key,
  3  d number (4) references x(a));
Table created.
SQL> create table z(
  2  e number (4) references x(b),
  3  f number (4) references y(c));
ERROR:no matching unique or primary key for this column-list
create table w(
e number (4) references x(a),
f number (4) references y(d));
ERROR no matching unique or primary key for this column-list
```

This is considered as a ‘foreign key’

Note: The table x must be created before we can create table y

```
SQL>  create table z(
  2  e number (4) references x(a),
  3  f number (4) references y(c));
Table created
```

Possible situations violating foreign key constraints:

Insert:

```
SQL> insert into y values(1, 2);
insert into y values(1, 2)
ERROR at line 1:ORA-02292: integrity constraint (SCOTT.SYS_C002908)
   violated - child record found
```

No tuple in x with primary key 2

Update:

Some tuple in y is referencing the current key value of a tuple x

```
SQL> update x set a=3;
ERROR at line 1:
ORA-02292: integrity constraint (SCOTT.SYS_C002908) violated - child record
found
SQL> update y set b=4;
ERROR at line 1: ORA-02291: integrity constraint (SCOTT.SYS_C002908)
   violated - parent key not found
```

No tuple in x with primary key 2

Delete:

Some tuple in y is referencing the current key value of a tuple x

```
SQL> delete from x where a=2;
delete from x where a=2
ERROR at line 1: ORA-02292: integrity constraint (SCOTT.SYS_C002908)
   violated - child record found
```

Insert with null values is OK!

```
SQL> insert into y values(1,null);
1 row created.
SQL> insert into y values(2,null);
1 row created.

SQL> select * from y;
         C          D
---------- ----------
         1
         2
```

There are *three* policies choices for situations violating foreign key constraints

The **reject** policy *(default)*

The system will reject any such violating request and a run-time error will be generated. The database state will *not* change.

In case of update or delete request:

The **cascade** policy: changes to the referenced attributes are "mimicked" at the foreign key (e.g. y.b)

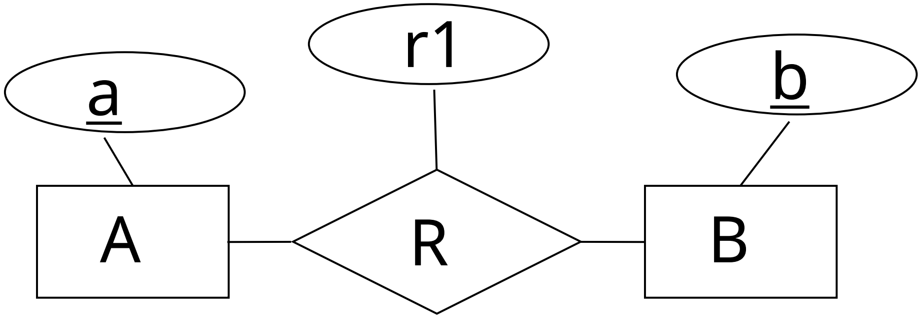
The **set-NULL** policy: set the referencing attribute to NULL (e.g., y.b)

Options/policies may be chosen for deletes and updates, *independently*

[ON DELETE {CASCADE | SET NULL }]

[ON UPDATE] { CASCADE | SET NULL }]

The policy to be used is a design decision and must conform to the business rules of the underlying application.



x_{pf} attribute is prime and foreign key

x_f attribute is foreign key

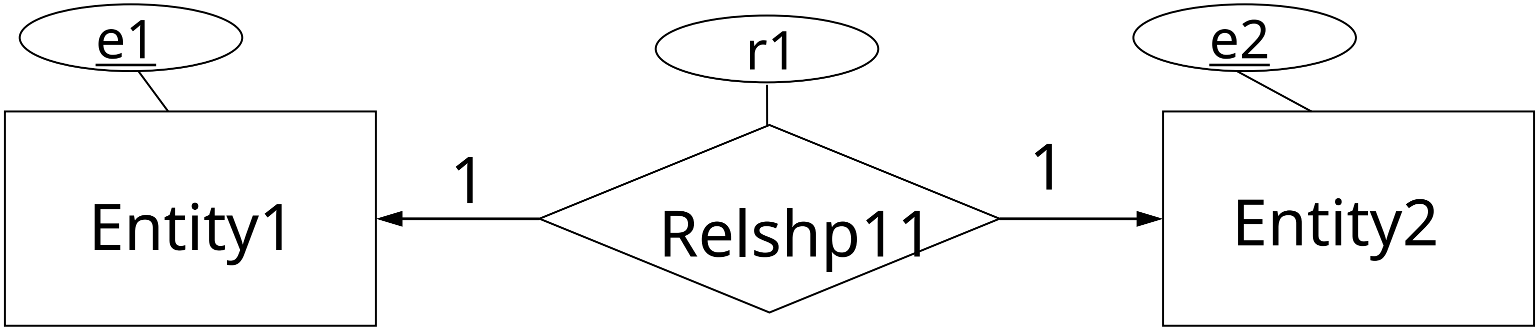
x_{fu} attribute is foreign key and unique

Attributes of R

Type of rel-ship	From A	From B	attrib
m-m	a_{pf}	b_{pf}	r
m-1	a_{pf}	b_f	r
1-1	a_{pf}	b_{fu}	r

For m-1 multiplicity either of the entities could be “A” – 2 cases

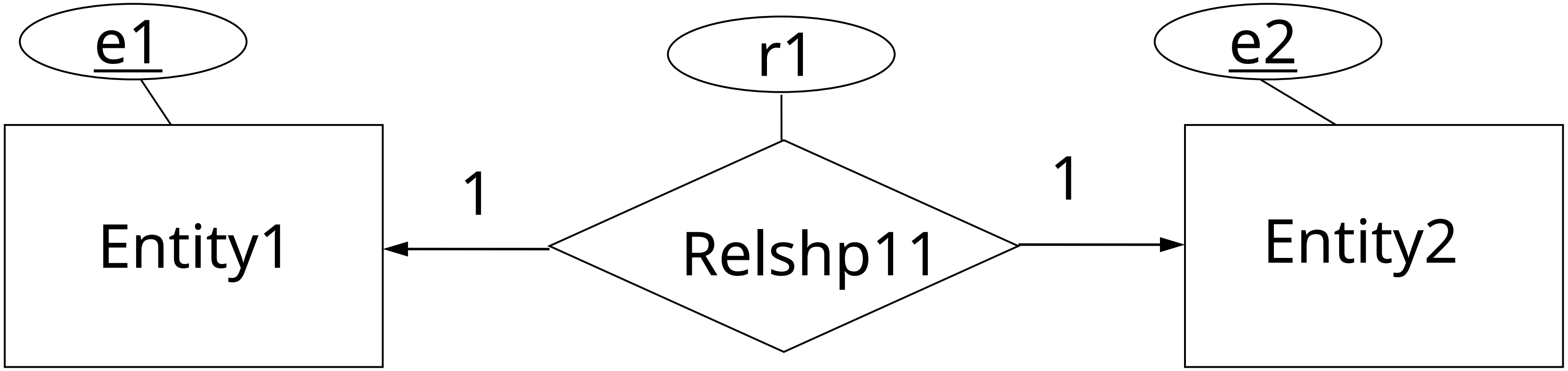
In the case of 1-1 multiplicity either of the entities could be “A” - 2 choices



Converting one to one relationship to a DB table:

```
create table RELSHP11
(e1 number primary key,
 e2 number,
 r1 number );
SQL> insert into RELSHP11 values (1, 1, 11);
1 row created.
SQL>insert into RELSHP11 values (1, 2, 11);
ERROR at line 1: ORA-00001: unique constraint violated
SQL> insert into RELSHP11 values (2, 1, 11);
1 row created. ←        Not a 1-to-1 relationship!
```

```
select * from relshp11;
        E1         E2         R1
---------- ---------- ----------
         1          1         11
         2          1         11
```

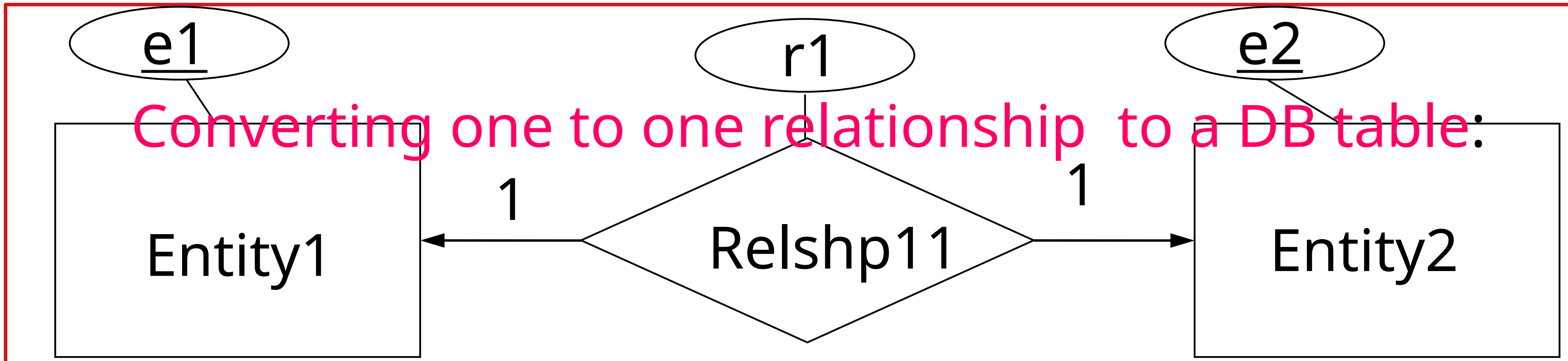


Converting one to one relationship to a DB table:

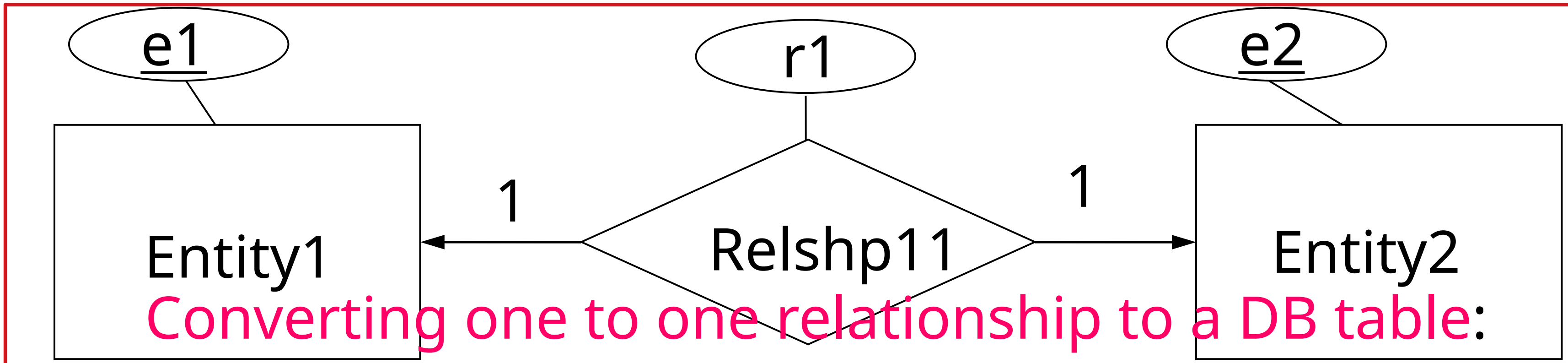
```
create table RELSHP11
(e1 number primary key,
 e2 number unique not null,
 r1 number );
SQL> insert into RELSHP11 values (1, 1, 11);
1 row created.
SQL>insert into RELSHP11 values (1, 2, 11);
ERROR at line 1: ORA-00001: unique constraint violated
SQL> insert into RELSHP11 values (2, 1, 11);
ERROR at line 1: ORA-00001: unique constraint violated
```

Without unique, we could insert the same e2 value for two different e1 values (not 1-to-1). Without null, we can leave out some values for entity e2. (not a 1-to-1 relationship!)

Converting one to one relationship to a DB table:



```
create table RELSHP11
(e1 number primary key, foreign key(e1) references ent1(e1),
 e2 number unique not null, foreign key(e2) references ent2(e2) ,
 r1 number);
SQL> insert into RELSHP11 values (1, 1, 11);
1 row created.
SQL>insert into RELSHP11 values (1, 2, 11);
ERROR at line 1: ORA-00001: unique constraint violated
SQL> insert into RELSHP11 values (2, 1, 11);
ERROR at line 1: ORA-00001: unique constraint violated
SQL> insert into relshp1m values(3,null,10);
*ERROR at line 1:ORA-01400: cannot insert NULL
```



Converting one to one relationship to a DB table:

```
SQL> create table e1(e1 number primary key);
Table created.
SQL> create table e2(e1 number primary key)
Table created.
create table r1(e1 number ,foreign key(e1) references e1(e1),
e2 number, foreign key(e2) references e2(e1),
r1 number, primary key (e1,e2))
insert into e1 values(1); insert into e1 values(2);
insert into e2 values(1) insert into e2 values(2);
insert into r1 values (1,1,10); insert into r1 values (2,1,10)
insert into r1 values (2,2,10);insert into r1 values (1,2,10)
```

WRONG design

```
create table r1new(e1 number unique ,foreign key(e1) references e1(e1),
e2 number unique, foreign key(e2) references e2(e1),
r1 number,
primary key (e1,e2) )
```

Added the unique attribute
With the unique attribute is
the primary key redundant?

```
insert into r1new values(1,1,10);
1 row created
insert into r1new values(2,1,10)
*
ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C004018) violated
```

Trying to insert another relationship involving e2.e1=1

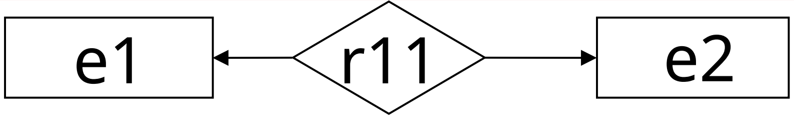
```
insert into r1new values(3,3,10)
*
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.SYS_C004020) violated - parent key
    not found
```

Referential integrity enforced

```
drop table e1
        *
ERROR at line 1:
ORA-02449: unique/primary keys in table referenced by foreign keys
```

Since the value(s) in e1 are being referenced

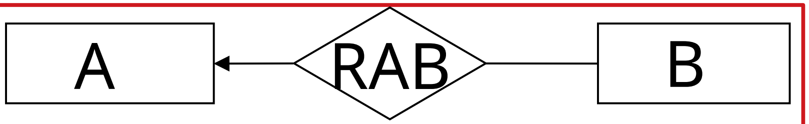
Mapping 1-to-1 relationship



Must create the referenced table (parent) before creating the referencing table (child)

```
create table e1(e1 number primary key);
create table e2(e1 number primary key);
create table r11(e1 number primary key,
foreign key(e1) references e1(e1) on delete cascade,
e2 number unique,
foreign key(e2) references e2(e1) on delete set null,
r1 number )
 insert into e1 values(1);    insert into e1 values(2);
insert into e2 values(1);     insert into e2 values(2);
insert into r11 values (1,1,10);  insert into r11 values (2,2,10);
 insert into r11 values (2,1,10);  UNIQUE CONSTRAINT
    VIOLATION
insert into r11 values (1,2,10);   UNIQUE CONSTRAINT
    VIOLATION
```

Mapping many-to-1 relationship



```
SQL> create table A(
a1 number(3) primary key,
a2 number (3)); Table created
```

```
SQL> create table B(
b1 number(4) primary key,
b2 number (3)); Table created
```

Must create the referenced table (parent) before creating the referencing table (child)

```
SQL> create table RAB(
  r1 number(3),
  r2 number(4) primary key,
  constraint fk_1 foreign key  (r1) references A(a1),
  constraint fk_2 foreign key  (r2) references B(b1) ); Table created.
SQL> insert into RAB values (null,null);
*ERROR ORA-01400: cannot insert NULL into ("SCOTT"."RAB"."R2")
SQL> insert into RAB values (null,1);
*ERROR ORA-02291: integrity constraint (SCOTT.FK_2) violated - parent key not found
SQL> insert into A values(1,1); 1 row created
SQL> insert into RAB values (null,1);
*ERROR ORA-02291: integrity constraint (SCOTT.FK_2) violated - parent key not found
SQL> insert into B values(11,11);1 row created.
SQL> insert into RAB values (1,12);
*ERROR ORA-02291: integrity constraint (SCOTT.FK_2) violated - parent key not found
SQL> insert into RAB values (2,11);
*ERROR ORA-02291: integrity constraint (SCOTT.FK_1) violated - parent key not found
SQL> insert into RAB values (1,11); 1 row created.
SQL> delete A where a1=1;
*ERROR ORA-02292: integrity constraint (SCOTT.FK_1) violated - child record found
SQL> delete B where b1=11;
*ERROR ORA-02292: integrity constraint (SCOTT.FK_2) violated - child record found
SQL> insert into A values (2, 2); 1 row created.
SQL> insert into RAB values (2,11);
* ERROR ORA-00001: unique constraint (SCOTT.SYS_C004197) violated
SQL> insert into B values (12, 12); 1 row created.
SQL> insert into RAB values (1, 12); 1 row created.
```

```
CREATE TABLE Supplier(
SID numeric(10) not null,
SName varchar2(50) not null,
Contact varchar2(50),
CONSTRAINT s_pk PRIMARY KEY (SID, SName));
```

```
CREATE TABLE Parts(
PNo numeric(10) not null,
SNo numeric(10) not null,
SName varchar2(50) not null,
CONSTRAINT p_fk
  FOREIGN KEY (SNo, SName)
 REFERENCES Supplier(SID, SName)  ON DELETE CASCADE);
```

The “ON CASCADE DELETE” in the foreign key constrain in Parts causes all tuples with the matching SID, SName values in Parts to be deleted when a record in Supplier is deleted

Added the on delete clauses

Changed the composite primary key

```
create table e1(e1 number primary key);
create table e2(e1 number primary key);
create table r11(e1 number primary key,
foreign key(e1) references e1(e1) on delete cascade,
e2 number unique,
foreign key(e2) references e2(e1) on delete set null,
r1 number )
 insert into e1 values(1);    insert into e1 values(2);
insert into e2 values(1);     insert into e2 values(2);
insert into r11 values (1,1,10);  insert into r11 values (2,2,10);
 insert into r11 values (2,1,10);  UNIQUE CONSTRAINT VIOLATION
insert into r11 values (1,2,10);   UNIQUE CONSTRAINT VIOLATION ;
```

```
SQL> select * from e1;
 E1
------
 1
2
```

```
SQL> select * from e2;
     E1
    -----
     1
     2
```

```
SQL> delete  from  e2;
2 rows deleted.
```

e2 columns set to null

```
SQL> select * from r11;
      E1         E2         R1
---------- ---------- ----------
      1                     10
      2                     10
```

```
SQL> delete from e1;
2 rows deleted.
```

Delete of rows in e1 cascades to r11

```
SQL> select * from r11;
no rows selected
```

```
create table e1(e1 number primary key);
create table e2(e1 number primary key);
create table r11(e1 number primary key,
foreign key(e1) references e1(e1) on delete cascade,
e2 number unique,
foreign key(e2) references e2(e1) on delete set null,
r1 number )
```

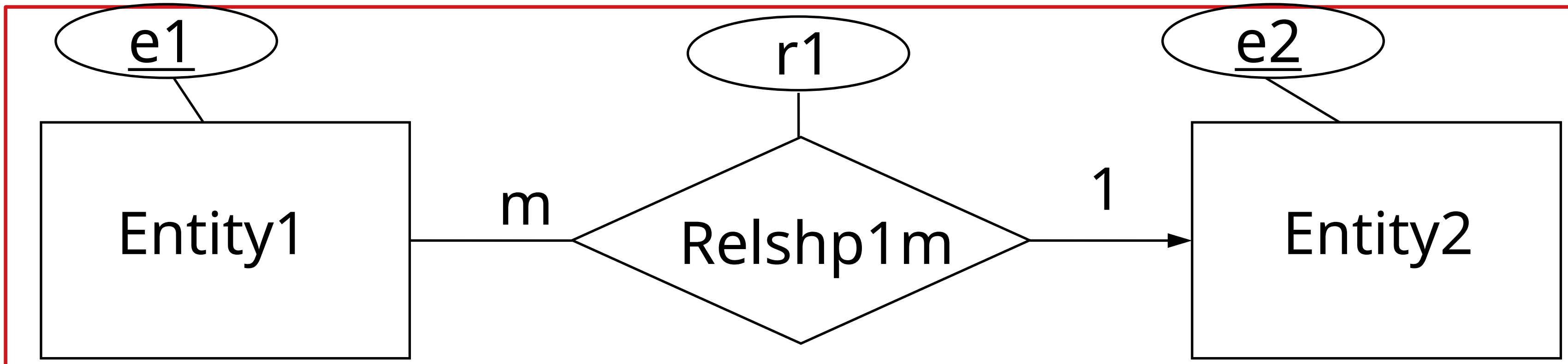
```
SQL> delete from e1;
x rows deleted
```

```
SQL> delete from e2;
y rows deleted
```

```
SQL> select * from r11;
no rows selected
```

```
SQL> insert into r11 values (3,null,null);
ERROR at line 1:
ORA-02291: integrity constraint (SCOTT.SYS_C004071)
violated - parent key not found
```

Can’t insert a null value in r11 of the foreign key constraint

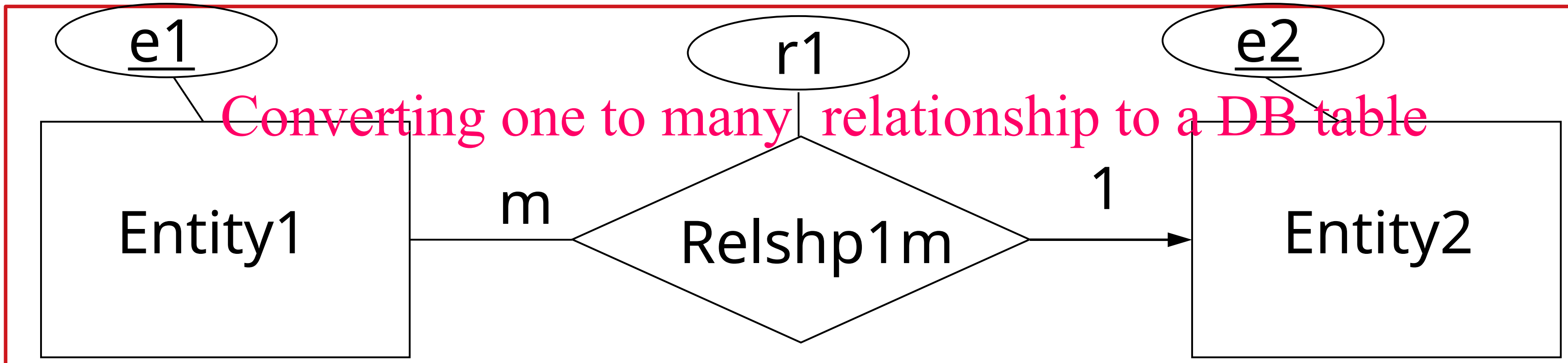


Converting one to many relationship to a DB table:

```
create table RELSHP1m
(e1 number primary key,
 e2 number,
 r1 number );
SQL> insert into relshp1m values (1, 1, 11);
1 row created.
SQL> insert into relshp1m values (1, 2, 11);
*ERROR at line 1:
ORA-00001: unique constraint violated
SQL> insert into relshp1m values (2, 1, 11);
1 row created.
```

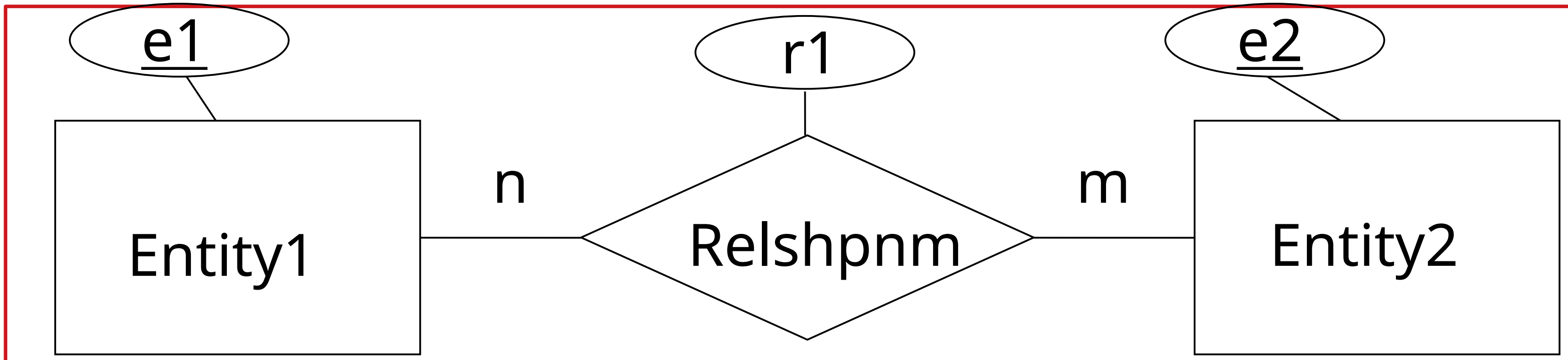
```
select * from relshp1m;

        E1         E2         R1
---------- ---------- ----------
         1          1         11
         2          1         11
```

Converting one to many relationship to a DB table



```
create table RELSHP1m
(e1 number primary key, foreign key(e1) references ent1(e1),
 e2 number not null, foreign key(e2) references ent2(e2),
 r1 number );
SQL> insert into relshp1m values (1, 1, 10);
1 row created.
SQL> insert into relshp1m values (1, 2, 10);
*ERROR 1: unique constraint violated
SQL> insert into relshp1m values (2, 1, 10);
1 row created.
SQL> insert into relshp1m values(2,3,10);
*ERROR integrity constraint violated - parent key not found
```

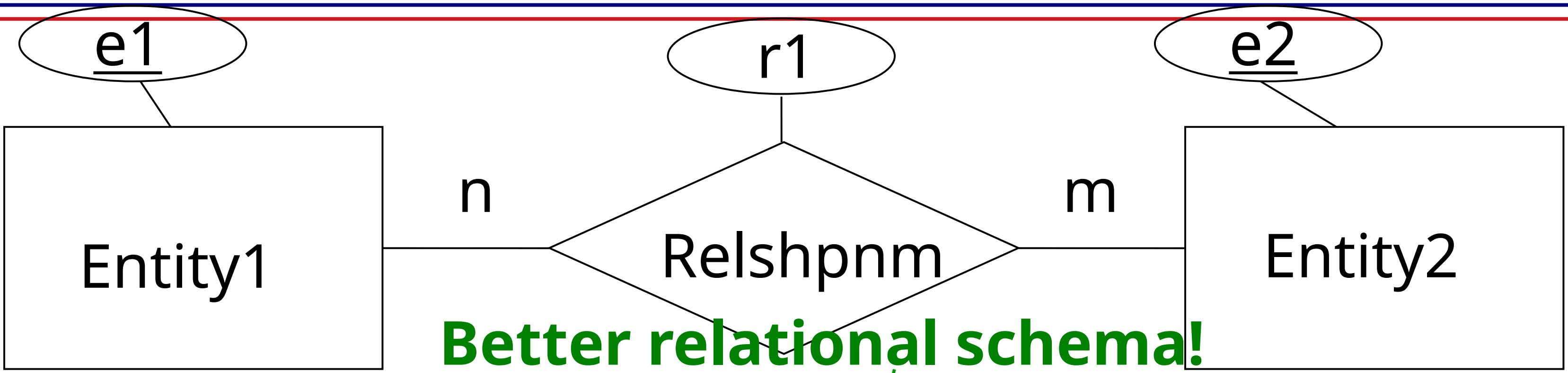
```
select * from relshp1m;
 E1         E2         R1
---------- ---------- ----------
         1          1         10
         2          1         10
```



Converting many to many relationship to a DB table:

```
create table RELSHPnm
(e1 number,  e2 number,
 r1 number, primary key(e1,e2) );
SQL> insert into relshpnm values (1, 1, 11)
1 row created.
SQL> insert into relshpnm  values (1, 2, 11);
1 row created.
SQL> insert into relshpnm  values (2, 2, 11);
1 row created.
SQL> insert into relshpnm  values (2, 1, 11)
1 row created.
```

```
select * from relshpnm;
        E1         E2         R1
---------- ---------- ----------
         1          1         11
         1          2         11
         2          2         11
         2          1         11
```



Better relational schema!

Converting many to many relationship to a DB table:

```
create table RELSHPnm
(e1 number,  e2 number,
 r1 number, primary key(e1,e2),
 foreign key(e1) references ent1(e1),
 foreign key(e2) references ent2(e2) );
SQL> insert into relshpnm values (1, 1, 11)
SQL> insert into relshpnm  values (1, 2, 11);
SQL> insert into relshpnm  values (2, 2, 11);
SQL> insert into relshpnm  values (2, 1, 11)
SQL> insert into relshpnm  values (3, 4, 11);
*ERROR: integrity constraint violated - parent key not found
```

```
select * from relshpnm;
        E1         E2         R1
---------- ---------- ----------
         1          1         11
         1          2         11
         2          2         11
         2          1         11
```

Alternate Candidate keys – How to implement?

Candidate keys of our friends:

Phone number

or

(name, address)

```
create table friends(
name varchar2(30) unique,
address varchar(30) unique,
phone decimal(16) primary key);
```

```
create table friends1(
name varchar2(30),
address varchar(30),
phone decimal(16) primary key,
 unique  (name,address));
```

```
create table friends2(
name varchar2(30),
address varchar(30),
phone decimal(16) unique,
primary key (name, address));
```

Alternate Candidate keys

```
create table friends (
name varchar2(30) unique,
address varchar(30) unique,
phone decimal(16) primary key);

SQL> insert into friends values('smith','montreal',1234);
1 row created.
SQL> insert into friends values('smith‘ ,'laval',1235);
ERROR : unique constraint (SCOTT.SYS_C003729) violated
SQL> insert into friends values('smith','montreal‘,1235);
ERROR : unique constraint (SCOTT.SYS_C003729) violated
SQL> insert into friends values('brown','laval',1235);
1 row created.
```

Only 1 Smith, 1 Montreal, etc.!

Alternate Candidate keys

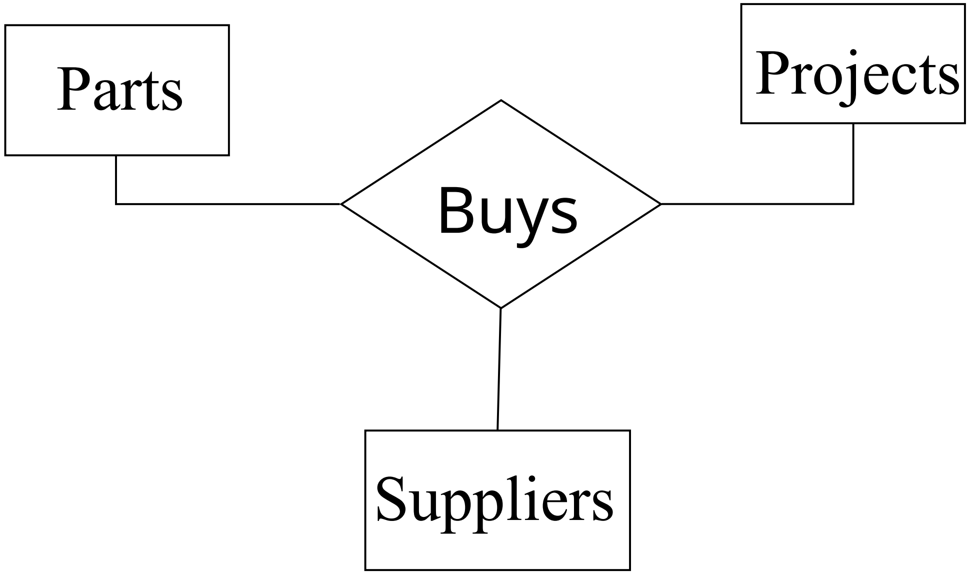
```
create table friends1(
name varchar2(30),
address varchar(30),
phone decimal(16) primary key,
 unique  (name,address));

SQL> insert into friends1 values('smith','montreal',1234);
1 row created.
SQL> insert into friends1 values('smith','laval',1236);
1 row created.
SQL> insert into friends1 values('smith','laval',1237)
* ERROR at line 1:
ERROR : unique constraint (SCOTT.SYS_C003727) violated
```

Alternate Candidate keys

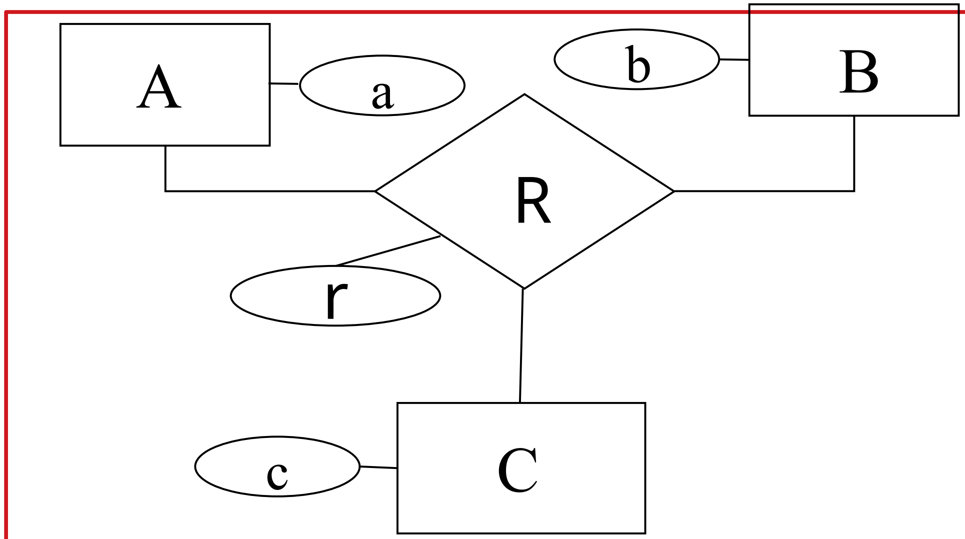
```
create table friends2(
name varchar2(30),  address varchar(30),
phone decimal(16) unique,  primary key(name,address));
SQL> insert into friends2 values('smith','laval',1235);
1 row created.
SQL> insert into friends2 values('smith','montreal',1235);
ERROR : unique constraint (SCOTT.SYS_C003724) violated
SQL> insert into friends2 values('brown','laval',1235);
ERROR : unique constraint (SCOTT.SYS_C003725) violated
SQL> insert into friends2 values('smith','laval‘,1236);
ERROR : unique constraint (SCOTT.SYS_C003724) violated
SQL> insert into friends2 values('smith','montreal',1234);
1 row created.
```

```
SQL> select * from friends2 /
NAME                    ADDRESS                     PHONE
------------------------- ------------------------------ ----------
smith                    montreal                   1234
smith                    laval                      1235
```

Ternary Relationships and multiplicity



The above is an example of a three way relationship
The multiplicity could be m or 1 for any of the entity sets involved in the relationship
Ignoring the permutation of the entities we need to consider
Four cases: m-m-m or m-m-1 or m-1-1 or 1-1-1
Considering permutations there are 8 cases 2*2*2 or 1+3+3+1



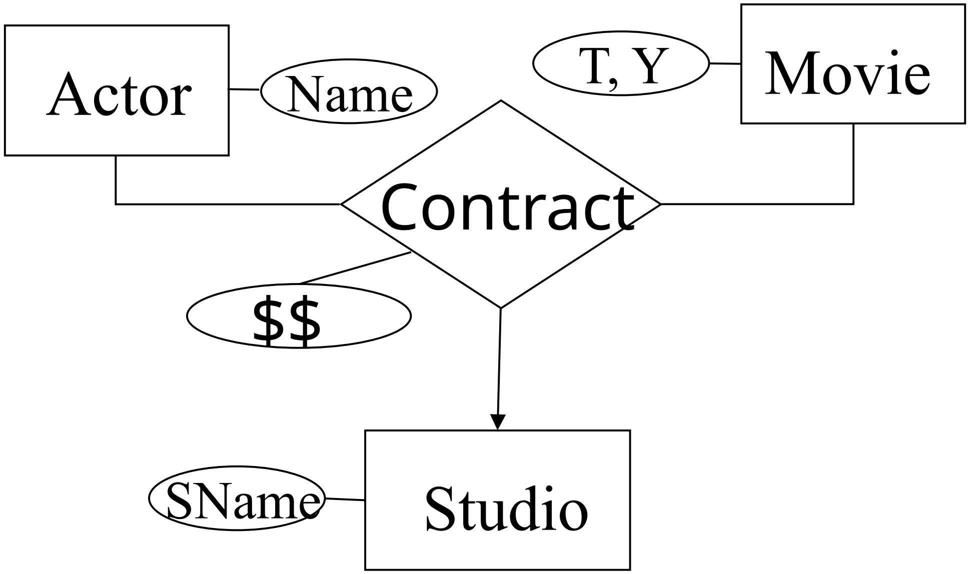
Type	From A	From B	From C	attrib
m-m-m	a_{pf}	b_{pf}	c_{pf}	r
m-m-1	a_{pf}	b_{pf}	c_f	r
m-1-1	a_{pf}	b_f	c_f	r
1-1-1	a_{pf}	b_{fu}	c_{fu}	r

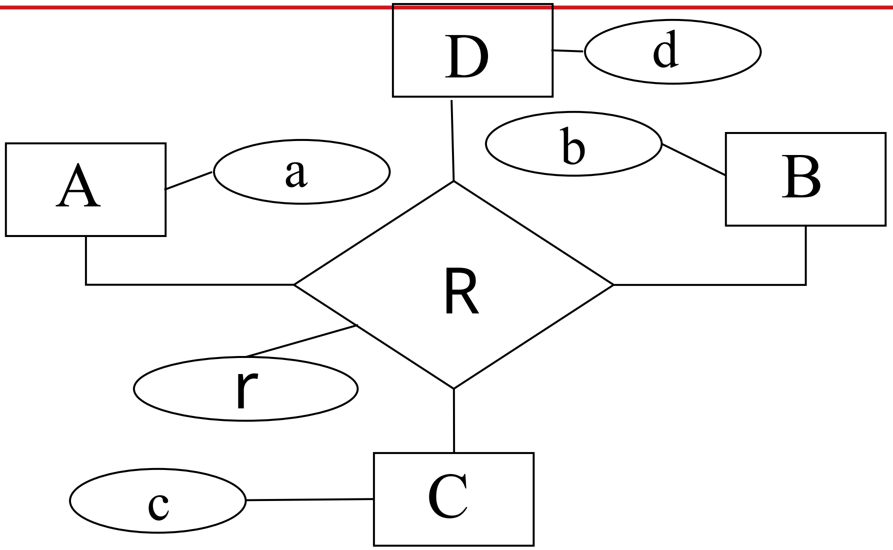
x_{pf} attribute is prime and foreign key

x_f attribute is foreign key

x_{fu} attribute is foreign key and unique

In the case of 1-1-1 multiplicity any of the entities could be used for A


Actor
Name
T, Y
Movie
Contract
$$
SName
Studio



x_{pf} attribute is prime and foreign key

x_f attribute is foreign key

x_{fu} attribute is foreign key and unique

Type	A	B	C	D	attrib
m-m-m-m	a_{pf}	b_{pf}	c_{pf}	d_{pf}	r
m-m-m-1	a_{pf}	b_{pf}	c_{pf}	d_f	r
m-m-1-1	a_{pf}	b_{pf}	c_f	d_f	r
m-1-1-1	a_{pf}	b_f	c_f	d_f	r
1-1-1-1	a_{pf}	b_{fu}	c_{fu}	d_{fu}	r

In the case of 1-1-1 multiplicity any of the entities could be used for A

Total # permutations would be 2*2*2*2= 1+4+6+4+1=16

Given the following relations, find the CS courses that Brenda can take.(Note: she cannot take a course already passed and must have all pre-requisites)

Student(Sno Name)
Dept(Dno, Dname)
Course(Cno, Dno, Cname)
Enroll(Sno,Cno,Grade)
Prereq(Cno,Pcno)

CSCrs = $\Pi_{\text{Cno}}\, \sigma_{\text{Dname=CS}}$ DEPT $\bowtie$ Course
BrendaPassed = Π_{Cno} (($\sigma_{\text{Grade} \neq \text{F}}$Enroll) $\bowtie$ ($\sigma_{\text{Name=Brenda}}$Student))
BrendaSatisfiedPre = CSCrs X BrendaPassed
BrendaCantTake = Π_{Cno} (Prereq - BrendaSatisfiedPre)
BrendaCanTake = (CSCrs – BrendaPassed) – BrendaCantTake

$\{c \mid c \in \text{Course} \wedge d \in \text{Dept} \wedge d.\text{Dname}=\text{‘COMP’} \wedge$

$s \in \text{Student} \wedge s[\text{Name}]=\text{‘Brenda’} \wedge c[\text{Dno}]=d[\text{Dno}] \wedge$

$\wedge \exists e(e \in \text{Enroll} \wedge e[\text{Sno}]=s[\text{Sno}] \wedge e[\text{Grade}]<>\text{’F’} \wedge$

$e[\text{Cno}]=C[\text{Cno}])$ Has not already taken and passed the course

$\wedge \forall p(p \in \text{Prereq} \wedge p[\text{Cno}] = C[\text{Cno}] \wedge p[\text{Cno}]=d[\text{Cno}]$

$\rightarrow \exists g(g \in \text{Enroll} \wedge s1 \in \text{Student} \wedge g[\text{Cno}] = p[\text{PCno}]$

$\wedge s1[\text{Name}]=\text{‘Brenda’} \wedge g[\mathit{Sno}] = s1[\text{Sno}]$

$\wedge g[\text{Grade}]<>\text{’F’} \}$

Has all the pre-req.

```
select c.cno
from course c, dept d, student s
where d.dept='Computer Science' and s.sname='Brenda' and
c.dno = d.dno and not exists(
select e.cno
from enroll e
where e.cno=c.cno
s.sno=e.sno and e.grade <>'F') and
not exists(select p1.cno
          from preq p1
          where p1.cno = c.cno and
          not exists(select e1.cno
                  from enroll e1, student s1
                    where e1.sno = s1.sno and  s1.sname='Brenda' and
                    e1.sno = s1.sno and  e1.grade <>'F' and
                    e1.cno = p1.pcno));
```

Has not already taken and
passed the course

Has all the pre-req.

Alternate SQL

```
select c.cno
from course c, dept d , student s
where d.dept='Computer Science' and s.sname='Brenda' and
c.dno = d.dno and c.cno not in(
select e.cno
from enroll e
where s.sno=e.sno and e.grade <>'F') and
not exists(select p1.cno
        from preq p1
        where p1.cno = c.cno and
        not exists(select e1.cno
                from enroll e1
                where e1.sno = s.sno and
                e1.sno = s.sno and  e1.grade <>'F' and
                e1.cno = p1.pcno));
```

The course c.cno has not already been taken and passed

Has all the pre-req.

We may associate the NOT NULL constraint with an attribute for a table

Two consequences:

1. We can’t insert a tuple into the table without giving value for the attribute defined with the NOT NULL constraint.

2. We can’t use the “set-null” policy to fix foreign-key violations for such attributes

```
SQL> create table z( d number (4)
          check (d >999));
Table created.
SQL> insert into z values(1);
insert into z values(1)
ERROR at line 1: ORA-02290: check constraint
   (SCOTT.SYS_C002909) violated
```

Difference between a check and a foreign-key constraint.

The check is done only when a tuple is inserted or updated.

A foreign key constraint checks for any update, deletes

- Example:

```
CREATE TABLE Star(
        name CHAR(30) PRIMARY KEY,
        address VARCHAR(255),
        gender CHAR(1),
         birthdate DATE,
        CHECK (gender = ’F’ OR name NOT LIKE ’Ms.%’)
 );
```

This constraints says that if a star is male (M), then his name must not begin with ‘Ms.’ (¬condition); Here we used (gender=‘F’ **OR** ¬condition) for (M→not Ms).

```
CREATE TABLE person(
        name CHAR(30) PRIMARY KEY,
        address VARCHAR(255),
        gender CHAR(1),
         dob DATE,
         CHECK (gender = 'F' OR name NOT LIKE 'Ms.%') );
        SQL> insert into person (name, gender)
            values ('Ms. John Smith', 'M');
        ERROR at line 1:
        ORA-02290: check constraint (SCOTT.SYS_C002916)
         violated
        SQL> insert into person (name, gender)
        values ('Ms. George Sands', 'F');
        1 row created.
        SQL> alter table person add constraint Person_Adr unique (gender);
        Table altered.
        SQL> alter table person add income number (12,2);
        Table altered.
```

Assertions, or *general constraints*, are boolean-valued SQL expressions that must *always* be true

Sometimes we need a constraint that involves relation as a whole or part of the database schema

Assertions are checked when a mentioned relation changes

Assertion in SQL not supported by most DBMS:

```
CREATE ASSERTION PoorPerson CHECK
      (NOT EXIST (SELECT *
                            FROM person
                           WHERE income > 10000
                  )
 );
```

Constraints and triggers

SQL provides a number of features to express *integrity constraints (primary, foreign key)*as part of the database schema.

Constraints, in essence, provide database designers with more control over the database content

An *active* element is an statement that we write once, store in the database, and “program” to execute when an event occurs. This event is considered as a **trigger**

An event may be an insertion of a tuple into a predefined table or a specified change in the database that causes a specified (boolean-valued) condition to become true

It is also possible to implement many of the constraints and triggers in scripts such as PHP, JSP etc. However, this is left to the application programmer and has to be included in each application!

Triggers

Procedures that are ***implicitly*** executed when an INSERT, UPDATE, or DELETE statement is issued against an associated table.

Simple procedure is ***explicitly*** executed by a user, application, or a trigger.
Triggers (one or more) are implicitly executed by Oracle when a triggering INSERT, UPDATE, or DELETE statement is issued, regardless of how it is issued(user or application).

A trigger can restrict DML operations against a table
(time of day/week etc.)

A statement in a trigger body could causes another trigger to be fired! Such the triggers are said to be *cascading triggers*.

Why Triggers?

- A required referential integrity rule cannot be enforced using: the integrity constraints such as:
  NOT NULL,
  UNIQUE key,
  PRIMARY KEY,
  FOREIGN KEY,
  CHECK,
  update CASCADE,
  update and delete SET NULL,
  update and delete SET DEFAULT
  - Enforce referential integrity when child and parent tables are on different nodes of a distributed database
  - Enforce complex business rules not definable using integrity constraints

A trigger has three parts:

- a triggering ***event*** (some statement)
- a trigger ***condition***
- a trigger ***action***

TRIGGERS

Triggers are often called **event-condition-action** rules

- An ***Event*** is any specified changes in the DB, due to insertions, deletions or updates
- *A **Condition*** is a predicate or test to determine if the specified trigger is applicable
- *An Action* is one or more SQL statements

Triggers are not supported in SQL2

Differ from checks or SQL2 assertions in that:

Event is programmable, rather than **implied** by the kind of check

Condition is not available in checks

Action could be any sequence of database operations

Triggers are essential for Active Database Management Systems (ADBMS)

Triggers are compiled by storing the procedure in a text file and compiling it with:

@filename

If we update an entire table with an SQL statement

- A **row-level trigger** will be executed once for each tuple
- A **statement-level trigger** will be executed only once for the entire update

In a statement-level trigger:

- We can not refer to old and new tuples
- Instead, we can/should refer to
  - The set of old tuples – **OLD TABLE**
  - The set of new tuples – **NEW TABLE**

Relation scheme: Employee(name, empId, salary, dept, supervisorId)

Constraint: *No employee gets a salary more than his/her supervisor.*

```
CREATE TRIGGER Inform_supervisor
BEFORE INSERT OR UPDATE OF salary, supervisorId ON Employee
NEW ROW AS new
FOR EACH ROW
WHEN (new.salary > (SELECT salary
                    FROM Employee
                    WHERE empId=new.supervisorId))
Begin
   ROLLBACK;
   Inform_Supervisor(new.supervisorId, new.empId);
End;
```

Row Triggers (before and after)

A *row trigger* is fired once for **each** row affected by, say, an UPDATE statement.

Row triggers are used when the trigger action depends on data provided by the triggering statement or rows that are affected.

Statement Triggers (before and after)

A *statement trigger* is fired **once**, regardless of the number of rows in the table that the triggering statement affects (even if no rows are affected).

If a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once, regardless of the number of rows are deleted from the table.
Useful when the trigger action does not depend on the data provided by the triggering statement or the rows that are affected.

Relation Scheme: Movie(title, year, length, filmType, studioName, producerC#)

```
CREATE VIEW ParamountMovie AS
SELECT title, year
FROM Movie
WHERE studioName = ’Paramount’;
```

The following *trigger* replaces an insertion on the view (ParamountMovie) with an insertion on its underlying base table (Movie)

```
CREATE TRIGGER ParamountInsert
INSTEAD OF INSERT ON ParamountMovie
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
INSERT INTO Movie(title, year, studioName)
VALUES (NewRow.title, NewRow.year, ’Paramount’);
```

This is WRONG for a view!

```
CREATE or REPLACE TRIGGER ParamountInsert
INSTEAD OF INSERT ON ParamountMovie
FOR EACH ROW
BEGIN
INSERT INTO Movie(title, year, studioName)
VALUES (:new.title, :new.year, 'Paramount');
end ParamountInsert;
/
Trigger created.
SQL> show errors trigger ParamountInsert;
No errors.
insert into ParamountMovie values ('Movie2016', '2016');
1 row created.
SQL> select * from movie;
TITLE                                YEAR   STUDIONAME
------------------------------ ----  ------------
Movie2016                             2016  Paramount
```

To-date Mysql/ Maraiadb doesn't have 'instead of insert' option

```
SQL> desc student;
 Name           Null?    Type
 -------------- -------- --------------
 SID            NOT NULL NUMBER(7)
 SNAME                   VARCHAR2(20)
 MAJOR                   CHAR(4)
 YEAR                    NUMBER(1)
 BDATE                   DATE

create view  cstdnt as
select sid as id, sname  as name, bdate as dob
from student
where major='COMP';
View created.

SQL> select * from cstdnt;
        ID NAME                 DOB
---------- -------------------- ---------
         8 Brenda               13-AUG-89
```

```
CREATE OR REPLACE TRIGGER CStudentInsert
INSTEAD OF INSERT ON cstdnt
REFERENCING NEW AS NewRow
FOR EACH ROW
INSERT INTO student(sid, sname, major, year, bdate)
VALUES (NewRow.id, NewRow.name, 'COMP', 1, NewRow.dob)
Warning: Trigger created with compilation errors.

SQL> SHOW ERRORS TRIGGER CStudentInsert;
Errors for TRIGGER CSTUDENTINSERT:

LINE/COL ERROR
-------- -----------------------------------------------------
1/7      PL/SQL: SQL Statement ignored
2/51      PL/SQL: ORA-00984: column not allowed here
```

```
SQL> CREATE OR REPLACE TRIGGER CStudentInsert
instead of INSERT ON cstdnt
FOR EACH ROW
INSERT INTO student(sid, sname, major, year, bdate)
VALUES (:new.id, :new.name, 'COMP', 1, :new.dob)
/
Trigger created.
SQL> select * from student;

    SID SNAME        MAJO   YEAR BDATE
------- ------------ ---- ------ ---------
      8 Brenda       COMP      2 13-AUG-89
     10 Dupont       ENGL      1 13-MAY-80
     13 Kelly        SENG      4 12-AUG-80
     14 Jack         CSAP      1 12-FEB-77
```

```
SQL> insert into cstdnt values(7,'Drew', '13-Sep-81');
1 row created.

SQL> select * from student;


   SID SNAME       MAJO    YEAR BDATE
------ ----------- ---- ------- ---------
     8 Brenda      COMP       2 13-AUG-89
    10 Dupont      ENGL       1 13-MAY-80
    13 Kelly       SENG       4 12-AUG-80
    14 Jack        CSAP       1 12-FEB-77
     7 Drew        COMP       1 13-SEP-81
```

```
MariaDB [tempDB]> CREATE TABLE users
( userid INT(11) NOT NULL AUTO_INCREMENT,
  last_name  VARCHAR(30) NOT NULL,
  first_name VARCHAR(25),
  email  VARCHAR(50)  NOT NULL,
  insert_date DATE,
  inserted_by VARCHAR(30),
  CONSTRAINT users_pk PRIMARY KEY (userid) );
```

```
MariaDB [mysql]> desc users;
+-------------+-------------+------+-----+---------+----------------+
| Field       | Type        | Null | Key | Default | Extra          |
+-------------+-------------+------+-----+---------+----------------+
| userid      | int(11)     | NO   | PRI | NULL    | auto_increment |
| last_name   | varchar(30) | NO   |     | NULL    |                |
| first_name  | varchar(25) | YES  |     | NULL    |                |
| email       | varchar(50) | NO   |     | NULL    |                |
| insert_date | date        | YES  |     | NULL    |                |
| inserted_by | varchar(30) | YES  |     | NULL    |                |
+-------------+-------------+------+-----+---------+----------------+
6 rows in set (0.01 sec)
```

```
DELIMITER |
CREATE TRIGGER users_before_insert
BEFORE INSERT
   ON users FOR EACH ROW
BEGIN
   DECLARE whoinserted varchar(50);
   -- Find username of person performing inserting a new user
   SELECT USER() INTO whoinserted ;
   -- Update create_date field to current system date
   SET NEW.insert_date = SYSDATE();
   -- Update created_by field to the username of the person
   --        performing the INSERT
   SET NEW.inserted_by = whoinserted;
END; |

DELIMITER ;
```

Other triggers:
- -before delete
- -before update
- -after insert
- -after update
- -after delete

```
show triggers;
```

```
MariaDB [test]> insert into users( last_name,  first_name, email)
     ->  values ('Smith', 'John', 'j_smith@okkefeenukee.edu');
Query OK, 1 row affected (0.024 sec)
```

Now create trigger

```
MariaDB [test]> DELIMITER |
MariaDB [test]> CREATE TRIGGER users_before_insert
    -> BEFORE INSERT
    ->    ON users FOR EACH ROW
    -> BEGIN
    ->    DECLARE whoinserted varchar(50);
    ->    -- Find username of person performing inserting a new user
    ->    SELECT USER() INTO whoinserted ;
    ->    -- Update create_date field to current system date
    ->    SET NEW.insert_date = SYSDATE();
    ->    -- Update created_by field to the username of the person
    ->    -- performing the INSERT
    ->    SET NEW.inserted_by = whoinserted;
    -> END; |
Query OK, 0 rows affected (0.011 sec)

MariaDB [test]> DELIMITER ;
```

Insert another user

```
MariaDB [tempDB]>  insert into users( last_name,  first_name,
email) values ('Smith', 'John', 'j_smith@okkefeenukee.edu');
Query OK, 1 row affected (0.01 sec)

MariaDB [tempDB]> select * from users;
select * from users;
+--------+-----------+------------+--------------------------+-------------+-------------------+
| userid | last_name | first_name | email                    | insert_date | inserted_by       |
+--------+-----------+------------+--------------------------+-------------+-------------------+
|      1 | Smith     | John       | j_smith@okkefeenukee.edu | NULL        | NULL              |
|      2 | Smith     | John       | j_smith@okkefeenukee.edu | 2024-03-24  | bcdesai@localhost |
+--------+-----------+------------+--------------------------+-------------+-------------------+
2 rows in set (0.000 sec)
```

Before the trigger was crated

After the trigger was crated

HOW ARE TRIGGERS EXECUTED

- SQL statement is issued
- Execute any BEFORE statement-level triggers
- For each row affected by the triggering SQL statement
    - Execute any BEFORE row-level triggers
    - Lock and change row, and perform integrity constraint checking
      The lock is not released until the transaction is commited
    - Execute any AFTER row-level triggers
- Execute any AFTER statement-level triggers

Example with triggers etc.

```
create table University(
 Name CHAR(20) PRIMARY KEY,
 City  CHAR(20));

create table Engineer(
 EID NUMBER(4),
 SIN NUMBER(9),
 Name char(20),
 AlmaMater CHAR(20),
 HireAge number(2) CHECK (HireAge BETWEEN 25 AND 65),
 CONSTRAINT Engineer_PK PRIMARY KEY(eid),
 CONSTRAINT Engineer_CK UNIQUE(SIN),
 FOREIGN KEY (AlmaMater) REFERENCES University(NAME)
 );
```

Name of the constraint

Candidate key

Check constraint

```
create table Project(
 ProjNo NUMBER(4) primary key,
 EID  NUMBER(4),
 FOREIGN KEY (EID) REFERENCES Engineer(EID)
);

create table Assigned(
  ID NUMBER(4),
  pno NUMBER(4),
  CONSTRAINT Assign_FK1 FOREIGN KEY(ID)
        REFERENCES Engineer(EID),
  CONSTRAINT Assign_FK2 FOREIGN KEY(pno)
        REFERENCES Project(ProjNo)
);
```

INSERT Some Data

```
insert into University  values('ConU', 'Montreal');
insert into University  values('UdeM', 'Montreal');

insert into  Engineer values(11, 123456789, 'Smith', 'ConU', 35);
insert into  Engineer values(12, 234567891, 'Shah', 'UdeM', 73)
*
ERROR at line 1:
ORA-02290: check constraint (SCOTT.SYS_C003385) violated

insert into  Engineer values(12, 234567891, 'Shah', 'UdeM', 33);
insert into  project values(1, 11);
insert into  project values(2, 11);
insert into  project values(3, 11);
insert into   Assigned values(11, 1);
insert into  Assigned values(12, 1);
```

```
SQL> Create or Replace package ProjEngg
  as
  EnggEid number(4);
  end;
/
Package created.
```

Package is a collections of procedures and functions

Trigger of type Row level

```
SQL> Create or Replace Trigger WhichEngg
Before Insert on Project
for each row
begin
ProjEngg.EnggEid  := :new.EID;
end;
/
Trigger created.
```

Statement level trigger

```
Create or Replace Trigger NumberOfProjs
After Insert on PROJECT
Declare Howmany Number(2);
Begin
Select count(ProjNo) into Howmany
from PROJECT
where EID = ProjEngg.EnggEid;
if ( Howmany > 4) then
RAISE_APPLICATION_ERROR(-20001,
                    '**** Too many projects for this engineer! ****');
end if;
end;
/
Trigger created.
```

```
SQL> insert into  project values(4, 11);
1 row created.
SQL> select * from project;
  PROJNO          EID
---------- ----------
         1         11
         2         11
         3         11
         4         11
```

```
SQL> select * from project;
  PROJNO          EID
---------- ----------
         1         11
         2         11
         3         11
         4         11
```

```
SQL> insert into  project values(5, 11);
ERROR at line 1:
ORA-20001: **** Too many projects for this engineer! ****
ORA-06512: at "SCOTT.NUMBEROFPROJS", line 7
ORA-04088: error during execution of trigger
            'SCOTT.NUMBEROFPROJS'
```

Mutating trigger

A trigger that attempts to modify the same table that initiated the trigger is called a mutating trigger.

```
CREATE OR REPLACE TRIGGER person_st
AFTER INSERT ON PERSON
REFERENCING NEW AS newRow
FOR EACH ROW
DECLARE STATUS CHAR(4);
BEGIN
STATUS := 'Poor';
IF (:newRow.income > 20000) THEN
STATUS := 'Med'; END IF;
IF (:newRow.income > 60000)THEN
STATUS := 'Rich'; END IF;
INSERT INTO person VALUES(:newRow.name, :newRow.dob,
           :newRow.income, STATUS);
END person_st;
.
run
```

Trigger created.

```
SQL> insert into person (name, dob,income)
        values('Jones', '10-jun-68', 61000.00);
insert into person (name, dob,income)
       values('Jones', '10-jun-68', 61000.00)
            *
ERROR at line 1:
ORA-04091: table SCOTT.PERSON is mutating,
        trigger/function may not see it
ORA-06512: at "SCOTT.PERSON_ST", line 11
ORA-04088: error during execution of trigger
        'SCOTT.PERSON_ST'
Sql> drop trigger person_st;

Trigger dropped.
```

Getting around mutation!

```
create table person1 (
name char(25) primary key,
dob date,
income number(12,2))
/

drop table person cascade
constraints
/

create table person (
name char(25) primary key,
dob date,
income number(12,2),
status char (6))
/
```

```
CREATE or REPLACE TRIGGER
person1_st
AFTER INSERT ON PERSON1
REFERENCING NEW AS newRow
FOR EACH ROW
DECLARE STATUS CHAR(4);
BEGIN
STATUS := 'Poor';
IF (:newRow.income > 20000) THEN
STATUS := 'Med';  END IF;
IF (:newRow.income > 60000)THEN
STATUS := 'Rich';  END IF;
INSERT INTO person VALUES
(:newRow.name, :newRow.dob,
:newRow.income, STATUS);
END person_st;
.
run
```

```
SQL> insert into person1 values('Smith', '31-may-70', 21000.00);
SQL> insert into person1 values('John', '3-Apr-68', 11000.00);
SQL> insert into person1 values('Wang', '31-may-70', 60001.00);
SQL> select * from person1;
NAME              DOB           INCOME
--------------- --------- ----------
Smith                 31-MAY-70      21000
John             03-APR-68      11000
Wang             31-MAY-70      60001
SQL> select * from person;
NAME              DOB        INCOME    STATUS
-------------------------- ---------- ------
Smith                 31-MAY-70      21000 Med
John             03-APR-68      11000 Poor
Wang             31-MAY-70      60001 Rich
```

Now input of dates needs the use of to_date function in Oracle!

```
insert into person1 values (‘Jones', to_date('1976/02/29','yyyy/mm/dd'), 29000.00);
```

```
SQL> select * from person;
NAME                      DOB           INCOME STATUS
------------------------- --------- ---------- ------
Smith                     31-MAY-70      21000 Med
John                      03-APR-68      11000 Poor
Wang                      31-MAY-70      60001 Rich

SQL> drop table person cascade constraints;
/
```

NOTE: Data in person1 is still not deleted

```
SQL> select * from person1;
NAME                      DOB           INCOME
------------------------- --------- ----------
Smith                     31-MAY-70      21000
John                      03-APR-68      11000
Wang                      31-MAY-70      60001
```

```
MariaDB [tempDB]>  create table person (
name char(25) primary key,
income decimal(12,2),
status char (6));
```

Note: the MySQL trigger syntax is different

```
CREATE OR REPLACE TRIGGER person_st
AFTER INSERT ON person
FOR EACH ROW
BEGIN
DECLARE STATUS CHAR(4);
set STATUS = 'Poor';
IF (new.income > 20000) THEN
set STATUS = 'Med';  END IF;
IF (new.income > 60000)THEN
set STATUS = 'Rich';  END IF;
INSERT INTO person VALUES(new.name, new.income, STATUS);
END
```

```
MariaDB [tempDB]>insert into person (name, income)
                                        values ('Smith', 10000.0);

ERROR 1442 (HY000): Can't update table 'person' in stored
 function/trigger because it is already used by statement
 which invoked this stored function/trigger.
```

Mutating Trigger in mariadb/mysql

Another way to get around mutation

```
SQL> create view personv as select * from person;
CREATE OR REPLACE TRIGGER personv_st
INSTEAD OF INSERT ON PERSONV
FOR EACH ROW
DECLARE STATUS CHAR(4);
BEGIN
STATUS := 'Poor';
IF (:new.income > 20000) THEN
STATUS := 'Med'; END IF;
IF (:new.income > 60000)THEN
STATUS := 'Rich'; END IF;
INSERT INTO person VALUES(:new.name, :new.dob,
           :new.income, STATUS);
END person_st;
.
run
Trigger created.
insert into personv values('Black', '13-may-45', 120000,'Poor');
1 row created.
```

```
SQL> select * from person;

NAME    DOB          INCOME STATUS
------ ---------- ------ ------
Smith   31-MAY-70   21000  Med
John    03-APR-68   11000  Poor
Wang    31-MAY-70   60001  Rich
Black   13-MAY-45   120000 Rich
```

```
SQL> select  TRIGGER_NAME from user_triggers;

TRIGGER_NAME
------------------------------
CSTUDENTINSERT
ECTRIG
NUMBEROFPROJS
PERSON1_ST
PERSONV_ST
WHICHENGG

6 rows selected.
```

```
insert into personv
(name, dob,income)
values('Jones', '10-jun-68', 61000);
1 row created.
```

```
SQL> select * from person;
NAME       DOB          INCOME STATUS
------- --------- ---------- ------
Smith   31-MAY-70       21000 Med
John    03-APR-68       11000 Poor
Wang    31-MAY-70       60001 Rich
Black   13-MAY-45      120000 Rich
Jones    10-JUN-68       61000 Rich
```

Another Mutating Trigger

```
create table EmpName (
     eid    number   not null,
     Name   varchar2(30),
     primary key(eid));
create table EmpCity (
     eid    number,
     City   varchar2(15),
     foreign key (eid) references EmpName(eid) on
   delete cascade);
insert into EmpName values(1,'Smith');
insert into EmpName values(2,'Lee');
insert into EmpCity values(1,'Montreal');
insert into EmpCity values(2,'Laval');
commit;
```

If an employee is deleted, his city is also deleted!

```
create or replace trigger ECTrig
    after delete on EmpCity
    for each row
    declare
        n integer;
begin
select count(*) into n from EmpName;
dbms_output.put _line ('There are ' || n ||' rows in EmpName');
dbms_output.put_line('after cascade delete of EmpCity');
dbms_output.new_line;
end;
.
run
Trigger created.
```

```
set serveroutput on;    To enable dbms_output
delete from EmpName where eid = 1;
            *
ERROR at line 1:
ORA-04091: table SCOTT.EMPNAME Is mutating,
trigger/function may not see it
ORA-06512: at "SCOTT.ECTRIG", line 4
ORA-04088: error during execution of trigger ‘
SCOTT.ECTRIG'
```

Solution: Use statement trigger instead of row trigger

```
create or replace trigger ECTrig
   after delete on EmpCity
   declare    n integer;
   begin
     select count(*) into n from EmpName;
     dbms_output.put ('There are  ' || n || '  rows in EmpName');
     dbms_output.put _line ('after cascade delete of EmpCity');
     dbms_output.new_line;
   end;
.
run
```

```
set serveroutput on;
delete from EmpName where eid = 1;
1 row deleted.
```

To enable dbms_output

```
There are  1  rows in EmpName
   after cascade delete of EmpCity
1 row deleted.
SQL> select * from EmpName;
EID   NAME
-----    -------------
   2     Lee
SQL> select * from EmpCity;
EID   CITY
------ --------------
    2    Laval
```

The mutating trigger error occurs due to the protocol used in Oracle to manage a **read consistent view** of data. (data read is of the **same** generation)

The error is occurs when a *row-level trigger*, while executing, accesses the table on which it is based.
The table is said to be mutating.

Mutation will not occur if a single record is inserted in the table (using VALUES clause).

If bulk insertion is done or data is inserted from another table mutation will occur.
The mutating error is not only encountered during queries, but also for insert, updates and deletes present in the trigger.

It is reported that newer release of the Oracle DBs (9i+) reduces the impact of the mutating triggers –but triggers still mutates.

Another example of Mutation

```
create table T  (A number, B varchar2(10));

SQL> create or replace trigger Ttrg
  2  before insert or update or delete
  3  on T
  4  for each row
  5  declare
  6    i pls_integer;
  7  begin
  8    select count(1)
  9    into   i
 10     from   T;
 11     dbms_output.put_line('Trigger success');
 12    exception
 13      when no_data_found then
 14      dbms_output.put_line('Error');
 15    end;
 16  /
Trigger created.
```

PLS_INTEGER

PLS_INTEGER instead of INTEGER or NUMBER for an efficient numeric datatype .

magnitude range for this datatype is –2147483647 through 2147483647.

require less storage than INTEGER or NUMBER values,

operations use faster machine arithmetic,

count(1) and count(*) returns the number of rows

```
SQL> insert into T values(1, 'ABD');
1 row created.
SQL> update T set A=2;            Bulk Update
update T
       *
ERROR at line 1:
ORA-04091: table SCOTT.T is mutating, trigger/function may not see it
ORA-06512: at "SCOTT.TTRG", line 4
ORA-04088: error during execution of trigger 'SCOTT.TTRG'
SQL> create table T1  (A number primary key, B varchar2(10));
SQL> insert into T1 values (1, ‘ABC');
SQL> insert into T1 values (1, ‘ABC');
SQL> insert into T select * from T1;       Bulk Insert
insert into T select * from T1
            *
                                T and T1 have the same schema
ERROR at line 1:
ORA-04091: table SCOTT.T is mutating, trigger/function may not see it
ORA-06512: at "SCOTT.TTRG", line 4
ORA-04088: error during execution of trigger 'SCOTT.TTRG'
```

Sln: Statement level trigger

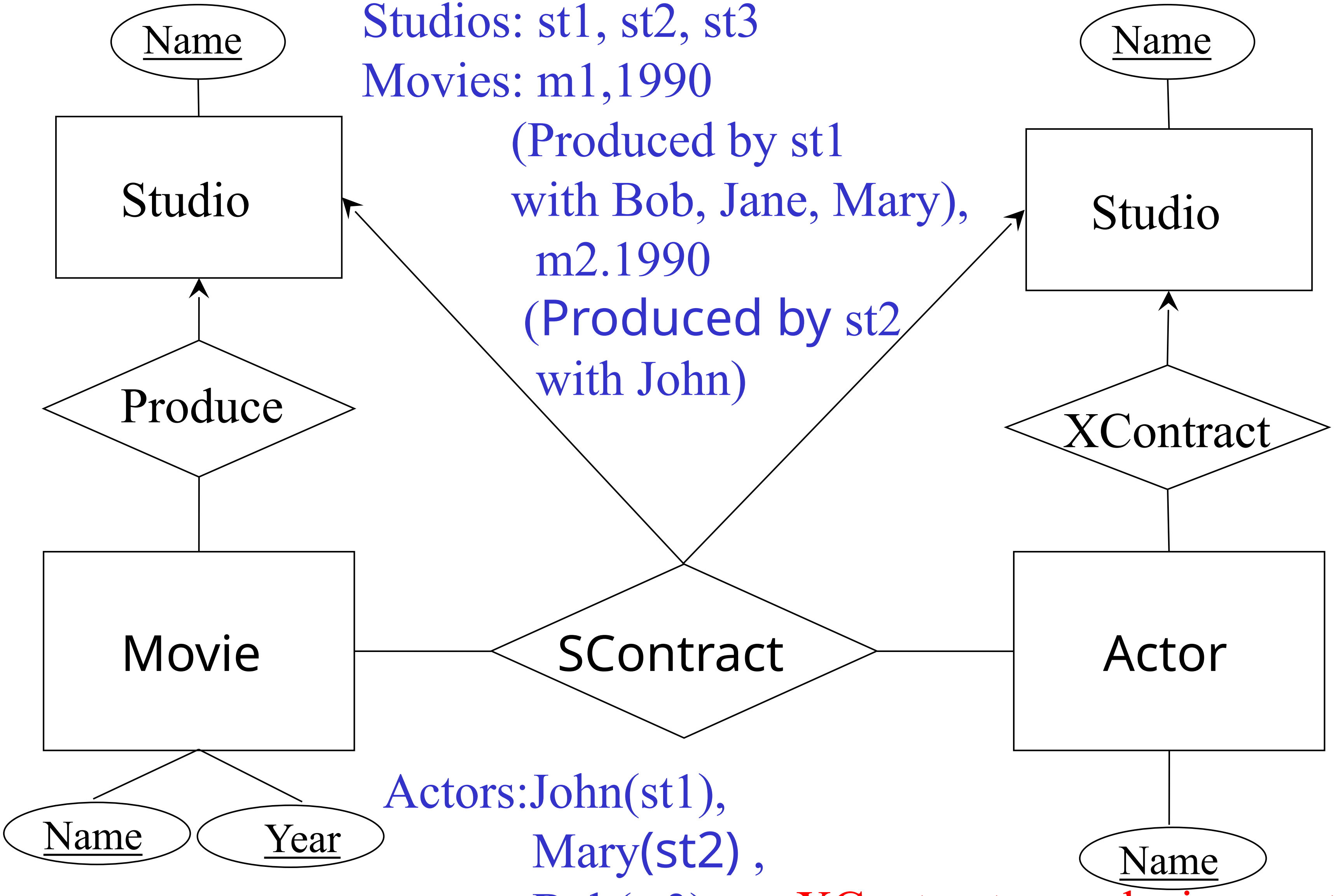
```
create or replace trigger Ttrg
before insert or update or delete on T
Declare   i pls_integer;
begin
  select count(1)   into   I   from   T;
  dbms_output.put_line('Trigger success');
 exception
   when no_data_found then
   dbms_output.put_line('Error');
 end;
SQL>  insert into T select * from T1;
2 rows created.
SQL> select * from T;

         A          B
---------- ----------
         1 ABD
         1 ABC
         2 BCD
```

Final example of Mutation

```
create table T1  (A number primary key, B varchar2(10));

create table T2 (A number, B  varchar2(10) ,
foreign key (A) references T1  on delete cascade);

create or replace trigger T1trg
before insert or update or delete on T1
for each row
Declare  i pls_integer;
begin
select 1
into   i
from   T2
where  A = :new.A;
dbms_output.put_line(‘Trigger success');
exception
when no_data_found then
dbms_output.put_line(‘Error: no data’);
end;
/
```

```
SQL> insert into T1 values (1, ‘ABC');
1 row created.
SQL> select * from T1;
         A          B
---------- ----------
         1 ABC

SQL> select * from t2;
no rows selected

SQL> delete from t1;
delete from t1
            *
ERROR at line 1:
ORA-04091: table SCOTT.T2 is mutating, trigger/function
may not see it
ORA-06512: at "SCOTT.T1TRG", line 4
ORA-04088: error during execution of trigger 'SCOTT.T1TRG'
```

```
SQL> insert into T1 values (2, ‘BCD');
1 row created.
SQL> insert into  T2 values(1, ‘XYZ');
1 row created.
SQL> insert into T2 values (2, ‘WXY');
1 row created.

SQL> delete from T1 where A= 1;
delete from T1 where A= 1
        *
ERROR at line 1:
ORA-04091: table SCOTT.T2 is mutating, trigger/function
may not see it
ORA-06512: at "SCOTT.T1TRG", line 4
ORA-04088: error during execution of trigger 'SCOTT.T1TRG'
```



Studios: st1, st2, st3
Movies: m1,1990
(Produced by st1
with Bob, Jane, Mary),
m2.1990
(Produced by st2
with John)

Actors:John(st1),
Mary(st2) ,
Bob(st3),
Jane(st3)

XContract – exclusive contract
SContract - special contract

```
create table studio(
name varchar2(12) primary key);
insert into studio values('st1');
insert into studio values('st2');
insert into studio values('st3');
```

```
create table actor(
name varchar2(12) primary key);
insert into actor values('John');
insert into actor values('Mary');
insert into actor values('Bob');
insert into actor values('Jane');
```

```
create table movie(
name varchar2(12), year dec(4),
primary key(name,year));
insert into movie values('m1',1990);
insert into movie values('m2',1990);
```

```
create table xcontract(
aname varchar2(12) primary key,
astudio  varchar2(12) not null,
foreign key (aname) references actor(name),
foreign key (astudio) references studio(name));
```

```
insert into xcontract
          values('John', 'st1');
insert into xcontract
          values('Mary', 'st2');
insert into xcontract
          values('Bob', 'st3');
insert into xcontract
          values('Jane', 'st3')
```

```
create table produce(
sname varchar2(12) not null unique,
mname  varchar2(12), myear  dec(4),
primary key (mname,myear),
-- must give the above  together as foreign key
foreign key (mname, myear ) references movie(name, year),
foreign key (sname) references studio(name));
insert into produce values('st1', 'm1',1990);
insert into produce values('st2', 'm2',1990);
```

```
create table scontract(  -- special contract
mname  varchar2(12), myear  dec(4),
aname  varchar2(12), pstudio varchar2(12),
astudio varchar2(12),
primary key(mname, myear, aname),
-- This is equivalent to a m-m-1-1 multiplicity
foreign key (mname,myear) references movie(name,year),
foreign key (aname) references actor(name),
foreign key (pstudio)  references studio(name),
foreign key (astudio)  references studio(name));

insert into scontract  values('m1',1990,'Mary',  'st1',   'st2');
insert into scontract  values('m1',1990,'Bob',    'st1',   'st3');
insert into scontract  values('m1',1990,'Jane',   'st1',   'st3');
insert into scontract  values('m1',1990,'Mary',  'st2',   'st1');
* ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C004729) violated
```

```
insert into scontract  values('m2',1990,'John',  'st2',   'st1');
 -- There is no check in consistency  John has exclusive contract
with st1
insert into scontract  values('m2',1990,'Bob',   'st2',   'st3');
 -- These is no check in consistency movie m2 in 1990 is made by   st2
  Bob is under contact to st3
insert into scontract  values('m2',1990,'Jane',  'st2',   'st3');
 -- These is no check in consistency movie m2 in 1990 is made by   st2
Jane is under contact to st3
```

```
SQL> select * from scontract;
MNAME             MYEAR ANAME       PSTUDIO     ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                 1990 Mary        st1         st2
m1                 1990 Bob         st1         st3
m1                 1990 Jane        st1         st3
m2                 1990 John        st2         st1
m2                 1990 Bob         st2         st3
m2                 1990 Jane        st2         st3
```

```
SQL> select * from xcontract;
ANAME        ASTUDIO
------------ ------------
Jane         st3
John         st1
Mary         st2
Bob          st3
```

Does not maintain the consistency

John is under contract to st1 **not** st3

Jane is under contract to st3 **not** st1

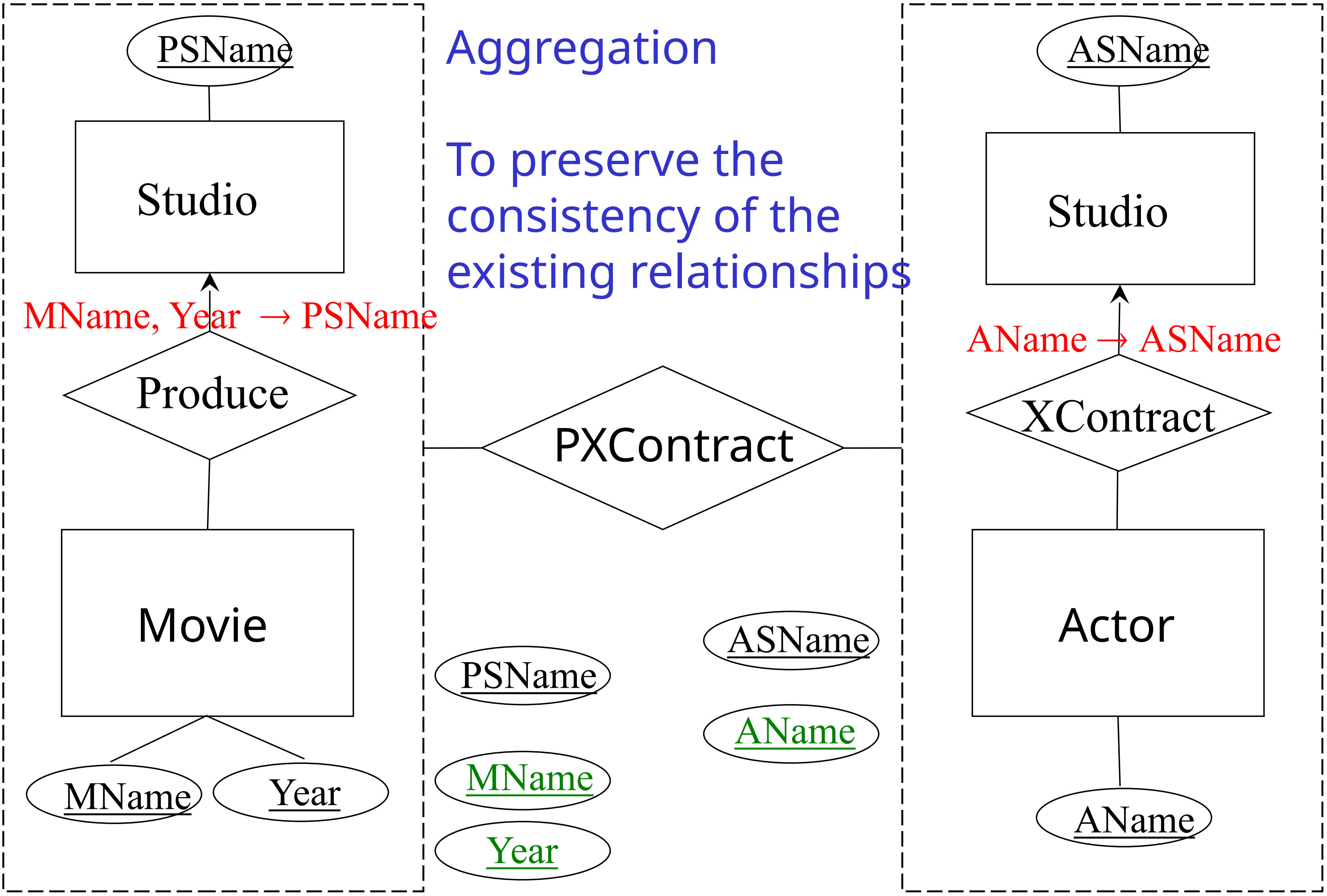
```
SQL> delete from scontract;
6 rows deleted.
insert into scontract values('m1',1990,'Jane','st1','st1');
insert into scontract values('m2',1990,'John','st2','st3');
SQL> select * from scontract;

MNAME           MYEAR ANAME      PSTUDIO    ASTUDIO
--------- -------- ---------- ---------- ----------
m1               1990 Jane       st1        st1
m2               1990 John       st2        st3
```

```
create table scontract1(
mname  varchar2(12),
myear  dec(4),
aname  varchar2(12),
pstudio varchar2(12),
astudio varchar2(12),
primary key(mname, myear, aname, pstudio, astudio),
-- This is equivalent to a 1 to 1 to 1 to 1 multiplicity
foreign key (mname,myear) references movie(name,year),
foreign key (aname) references actor(name),
foreign key (pstudio)  references studio(name),
foreign key (astudio)  references studio(name));
```

```
insert into scontract1  values('m1',1990,'Mary',  'st1',   'st2');
insert into scontract1  values('m1',1990,'Bob',   'st1',   'st3');
insert into scontract1  values('m1',1990,'Jane',  'st1',   'st3');
insert into scontract1  values('m2',1990,'John',  'st2',   'st1');
insert into scontract1  values('m2',1990,'Bob',   'st2',   'st3');
insert into scontract1  values('m2',1990,'Jane',  'st2',   'st3');

insert into scontract1  values('m1',1990,'Mary',  'st3',   'st1');
  --  Allows the same movie to be made by different studio
  --  Allows the same actor to be under contract to >1 studio!
```

Aggregation

To preserve the consistency of the existing relationships



```
drop table pxcontract;
create table pxcontract(
mname  varchar2(12),
year  dec(4),
pstudio varchar2(12),
aname  varchar2(12),
astudio varchar2(12),
primary key(pstudio, mname, year),
foreign key (mname,year) references produce(mname,myear),
foreign key (aname) references xcontract(aname));
SQL> insert into pxcontract  values('m1',1990, 'st1',  'Mary',  'st2');
1 row created.
SQL> insert into pxcontract  values('m1',1990, 'st1',  'Bob',    'st3');
* ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C004747) violated
```

Use triggers to maintain consistency

```
drop table pxcontract;
create table pxcontract(
mname  varchar2(12),
year  dec(4),
aname  varchar2(12),
primary key( mname, year, aname),
foreign key (mname,year)
        references produce(mname,myear),
foreign key (aname) references xcontract(aname));
```

USE TRIGGER for consistency

```
create view vcontract as select * from scontract;
CREATE OR REPLACE TRIGGER SP_Trig
Instead of INSERT ON vcontract
FOR EACH ROW
Declare pstudio varchar2(12);  astudio varchar2(12);
begin
select p.sname into pstudio from produce p
where :new.mname=p.mname
And :new.myear=p.myear;
select x.astudio into astudio from xcontract x
where :new.aname=x.aname;
INSERT INTO scontract values
(:new.mname,:new.myear, :new.aname,         pstudio, astudio);
END SP_Trig;
.
run
```

```
insert into vcontract  values('m1',1990,'Mary',  'st3',   'st1');
1 row created.

SQL> select * from scontract;

MNAME             MYEAR ANAME        PSTUDIO      ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                 1990 Mary         st1          st2

insert into vcontract  values('m1',1990,'Mary',  'st1',   'st2');

        * ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C004729) violated
ORA-06512: at "SCOTT.SP_TRIG", line 8
ORA-04088: error during execution of trigger 'SCOTT.SP_TRIG'
```

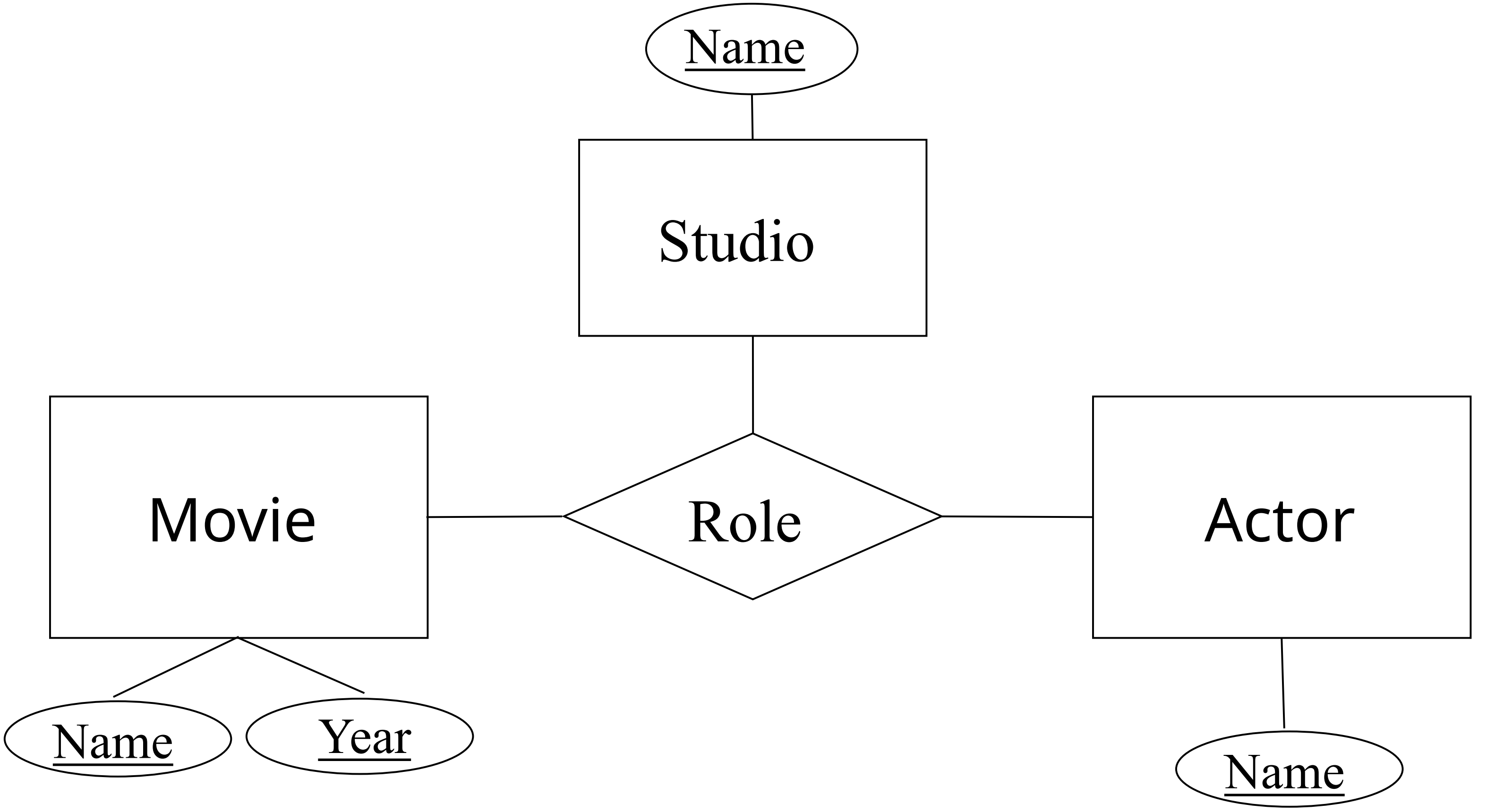
```
insert into vcontract  values('m1',1990,'Bob',   'st1',   'st3');
SQL>  select * from scontract;

MNAME             MYEAR ANAME        PSTUDIO      ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                 1990 Mary         st1          st2
m1                 1990 Bob          st1          st3
insert into vcontract  values('m1',1990,'Jane',  'st1',   'st3');
1 row created.

SQL> select * from scontract;

MNAME             MYEAR ANAME        PSTUDIO      ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                 1990 Mary         st1          st2
m1                 1990 Bob          st1          st3
m1                 1990 Jane         st1          st3
```

```
insert into vcontract  values('m2',1990,'John',  'st2',   'st1');
SQL> select * from scontract;
MNAME             MYEAR ANAME        PSTUDIO      ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                 1990 Mary         st1          st2
m1                 1990 Bob          st1          st3
m1                 1990 Jane         st1          st3
m2                 1990 John         st2          st1
```

```
insert into vcontract  values('m2',1990,'Bob',   'st2',   'st3');
SQL> select * from scontract;
MNAME             MYEAR ANAME        PSTUDIO      ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                 1990 Mary         st1          st2
m1                 1990 Bob          st1          st3
m1                 1990 Jane         st1          st3
m2                 1990 John         st2          st1
m2                 1990 Bob          st2          st3
```

```
insert into vcontract  values('m2',1990,'Jane',  'st2',   'st3');

SQL> select * from scontract;
MNAME                 MYEAR ANAME        PSTUDIO      ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                     1990 Mary         st1          st2
m1                     1990 Bob          st1          st3
m1                     1990 Jane         st1          st3
m2                     1990 John         st2          st1
m2                     1990 Bob          st2          st3
m2                     1990 Jane         st2          st3

SQL> insert into vcontract  values('m1',1990,'Mary',  'st3',   'st1')
            *ERROR at line 1:
ORA-00001: unique constraint (SCOTT.SYS_C004729) violated
ORA-06512: at "SCOTT.SP_TRIG", line 8
ORA-04088: error during execution of trigger 'SCOTT.SP_TRIG'
```

```
SQL> select * from scontract;
MNAME             MYEAR ANAME        PSTUDIO      ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                 1990 Mary         st1          st2
m1                 1990 Bob          st1          st3
m1                 1990 Jane         st1          st3
m2                 1990 John         st2          st1
m2                 1990 Bob          st2          st3
m2                 1990 Jane         st2          st3

6 rows selected.
SQL> select * from vcontract;
MNAME             MYEAR ANAME        PSTUDIO      ASTUDIO
------------ ---------- ------------ ------------ ------------
m1                 1990 Mary         st1          st2
m1                 1990 Bob          st1          st3
m1                 1990 Jane         st1          st3
m2                 1990 John         st2          st1
m2                 1990 Bob          st2          st3
m2                 1990 Jane         st2          st3
6 rows selected.
```



Free lance system where actors are not under contract

DATES:

How to specify the beginning weekday of the week

```
select to_char(trunc(sysdate,'DAY'),'fmDay"," Month DD"," YYYY')
                    AS First_week_day from dual;
FIRST_WEEK_DAY
--------------------------
Sunday, November 17, 2002
```

If we want Monday to be the beginning of the week:

```
SQL> alter session set nls_territory=FRANCE;
Session altered.
SQL> select to_char(trunc(sysdate,'DAY'),'fmDay"," Month DD",
        " YYYY')    AS First_week_day from dual;
FIRST_WEEK_DAY
-----------------------------
Monday, November 18, 2002
```

The DUAL table in Oracle

```
SQL> describe dual;
 Name                    Null?    Type
 ----------------------- -------- ----------------
 DUMMY                             VARCHAR2(1)
```

Contains one row and one column. Can be used to put results

```
SQL> select power(2,10) from dual;

POWER(2,10)
-----------
       1024
SQL> select to_date(sysdate) from dual;

TO_DATE(S
---------
29-SEP-02
```

```
SQL> select add_months(sysdate,2) from dual;
ADD_MONTH
---------
18-JAN-04
```

Lets make Brenda younger

```
SQL> select * from student where sid=8;
 SID   SNAME      MAJOR    YEAR      BDATE
-------- ------------- -------------- ---------    ----------------
       8  Brenda      COMP        2          13-AUG-77

 SQL> update student
set bdate=(select add_months(bdate,36)from dual)
where sid=8
SQL> select * from student where sid=8;

 SID SNAME               MAJOR YEAR BDATE
---- --------------- ----- ----
------------
   8 Brenda              COMP     2 13-AUG-80
```

TRUNCate function and dates

truncate to first day of week

```
select to_char(trunc(sysdate,'DAY'))  as FirstDayofWeek from dual;
FIRSTDAYO
----------
14-MAR-04
```

```
select to_char(trunc(sysdate)) from dual;
TO_CHAR(T
---------
16-MAR-04
```

Date of query (a Tuesday)

```
select to_char(trunc(sysdate,'DAY'),'fmDay') as FirstDay from dual;
FIRSTDAY
---------
Sunday
```

Format fully after truncating to first day of week, month, year

```
select to_char(trunc(sysdate,'MONTH'),'fmMonth') as Month from dual;
MONTH
----------
March
```

```
select to_char(trunc(sysdate,'MONTH')) from dual;
TO_CHAR(T
---------
01-MAR-04
```

```
select to_char(trunc(sysdate),'fmYear')  as year from dual;
YEAR
--------------------------
Two Thousand Four
```

```
select to_char(trunc(sysdate,'YEAR')) from dual;
TO_CHAR(T
---------
01-JAN-04
```

```
select to_char((select DOB from person where Name='Smith'),
            'DAY' ) As Weekday from dual;
WEEKDAY
---------
SUNDAY
```

To find the first business day of the week for a particular date:

```
SQL> select to_char(trunc((select DOB from person where
               Name='Smith'),'DAY') , 'fmDay') from dual;
Monday
```

To find the first business day of the week two days from a particular date:

```
SQL> select to_char(trunc((select DOB from person where
          Name='Smith'),'DAY') + 2, 'fmDay') from dual;
Wednesday
```

To find the day of the week for a specified date:

```
SQL> select to_char(to_date('18-Nov-02'), 'Day') As Weekday
          from dual;
WEEKDAY
---------
Monday
```

Find the first business day after the birthday of Smith:

```
select to_char(trunc((select DOB from person
                           where Name='Smith'),
                          'DAY') +2, 'fmDay') As
                         Smith_Bday2busWeek from dual;
```

Finding where a date is(half, quarter)

```
select TO_NUMBER(TO_CHAR((select DOB from person
              where Name='Smith'), 'Q')) as SmithQ from dual;
```

```
SmithQ
---------
2
```

DECODE function

DECODE(expression, if1, then1, if2, then2, …., ifn,thenn, else)

```
Create tables assignment(
 sid number(7),
 assign# number(2),
 submitdate date
 primary key (sid, assign#))
```

```
Create table duedate(
assign# number (2) primary key,
duedate date);
Insert into duedate
    values(1, ‘17-jan-2004’);
insert into duedate
    values(2, '14-feb-2004');
```

```
 insert into assignment values (123, 1, '17-jan-2004');
 insert into assignment values (124, 1, '18-jan-2004');
 insert into assignment values (125, 1, '19-jan-2004');
 insert into assignment values (123, 2, '17-feb-2004');
 insert into assignment values (124, 2, '18-feb-2004');
 insert into assignment values (125, 2, '19-feb-2004');
```

```
select a.sid, a.assigno,
  decode(trunc(a.submitdate - d.duedate), 0, null, 1, 'one day',
  2, 'two days', 3, 'three days', 4, 'four days' ,'Too late' ) as Late_days
from assignment a , duedate d
where a.assigno = d.assigno
order by sid, assigno;
```

```
 SID    ASSIGNO LATE_DAYS
-----   ------- ----------
  123         1
  123         2   three days
  124         1   one day
  124         2   four days
  125         1   two days
  125         2   Too late
```

To calculate the number of business days between two days: store the following code in a file say: numbusdays.sql

```
define frdate = '&1'
define todate = '&2'
set verify off
select
  '&frdate' From_Date,'&todate' To_Date,
   1 + to_date('&todate') - to_date('&frdate') -
    ((TRUNC(to_date('&todate'),'D') –
           TRUNC(to_date('&frdate'),'D'))/7)*2
    + DECODE(to_char(to_date('&todate'),'D'),7,-1,0)
    + DECODE(to_char(to_date('&frdate'),'D'),1,-1,0) Business_Days
 from dual
/
```

```
FROM_DATE TO_DATE   BUSINESS_DAYS
--------------- ------------------   -----------------
20-Nov-02       24-Dec-02         25
```

How many weeends?

a saturday?

a sunday?

DECODE (exp, if, then, else, ..)

Then one can interactively call it:

```
SQL> @numbusdays 20-Nov-02 24-Dec-02
SQL> create table interval (startdate char(10), enddate char(10));
insert into interval values('1998.04.11','1998.09.30');
insert into interval values('1998.04.15','1998.10.01');
insert into interval values('1998.05.11','1998.06.17');
insert into interval values('1998.06.14','1998.10.12');

SQL> SELECT STARTDATE,ENDDATE
FROM   INTERVAL
WHERE TO_DATE('1998.04.17','YYYY.MM.DD') BETWEEN
    TO_DATE(STARTDATE,'YYYY.MM.DD') AND
    TO_DATE(ENDDATE,'YYYY.MM.DD');

STARTDATE  ENDDATE
---------- ----------
1998.04.11 1998.09.30
1998.04.15 1998.10.01
```

If the interval is stored as dates:

```
SQL> create table intervaldate (startdate date, enddate date);
SQL> insert into intervaldate
select TO_DATE(startdate,'YYYY.MM.DD'),
    TO_DATE(enddate,'YYYY.MM.DD')
      from interval;
```

SQL> select * from intervaldate;

```
STARTDATE ENDDATE
--------- ---------
11-APR-98 30-SEP-98
15-APR-98 01-OCT-98
11-MAY-98 17-JUN-98
14-JUN-98 12-OCT-98
.
```

SQL> select startdate, enddate+1
from intervaldate;

```
STARTDATE ENDDATE+1
--------- ---------
11-APR-98 01-OCT-98
15-APR-98 02-OCT-98
11-MAY-98 18-JUN-98
14-JUN-98 13-OCT-98
```

```
SQL> SELECT startdate , enddate FROM   intervaldate
         WHERE TO_DATE('1998.07.03','YYYY.MM.DD')
           BETWEEN   startdate AND enddate ;
STARTDATE ENDDATE
--------- ---------
11-APR-98 30-SEP-98
15-APR-98 01-OCT-98
14-JUN-98 12-OCT-98
SQL> select TO_CHAR(startdate, 'YYYY-MM-DD:HH:MI:SS')
       as starttime      from intervaldate;
STARTTIME
-------------------
1998-04-11:12:00:00
1998-04-15:12:00:00
1998-05-11:12:00:00
1998-06-14:12:00:00
```

```
SQL> select TO_CHAR(startdate+
                    8/24 +    13/1440 + 12/86400,
                     'YYYY-MM-DD:HH:MI:SS' ) as NewStartTime
                      from intervaldate;
NEWSTARTTIME
-------------------
1998-04-11:08:13:12
1998-04-15:08:13:12
1998-05-11:08:13:12
1998-06-14:08:13:12
```

Last day of month:

```
SQL>select LAST_DAY(enddate)
       as EndofMonth
        from intervaldate;
```

```
ENDOFMONTH
---------
30-SEP-98
31-OCT-98
30-JUN-98
31-OCT-98
```

```
SQL> select Name,
        Trunc(MONTHS_BETWEEN(Sysdate, DOB)/12) as Age
         from person;
NAME    AGE
------ ---
Smith    32
John     34
Wang     32
```

Age in five years?

```
select Name,
 Trunc(MONTHS_BETWEEN(ADD_MONTHS(Sysdate,60),
         DOB)/12) as Age   from person;
```

```
NAME    AGE
------ ---
Smith    37
John     39
Wang     37
```

```
create or replace function time_between (start_tm in date, end_tm in date,
  hours_only varchar2 default 'N') return varchar2 as
-- If "hours_only" is null or "N", the return will be a string formatted like:
--     2 days, 3 hrs, 5 mins, 10 secs
-- If "hours_only" is not "N", then the return is a value in hours, like 102.325
 ret_val   varchar2(80);
 start_sec number;
 end_sec   number;
 full_sec  number;
 balance   number;
 minutes   number;
 hours     number;
 days      number;
--
 function get_sec (time_in in date) return number as
 begin
   return to_number(to_char(time_in,'SSSSS'));
 end;
--
begin
 start_sec := get_sec(start_tm);
 end_sec := get_sec(end_tm);
 -- check if end time is in the same day as start time
 if to_char(start_tm,'YYMMDD') =
        to_char(end_tm,'YYMMDD') then
   full_sec := end_sec - start_sec;
   days := 0;
 else
   days := trunc(end_tm - start_tm);
   if days > 0 then
     ret_val := to_char(days)||' days, ';
   end if;
   if end_sec > start_sec then
     full_sec := end_sec - start_sec;
   else
     full_sec := 86400 - start_sec + end_sec;
   end if;
 end if;
```

```
if upper(hours_only) = 'N' then
   if full_sec > 3599 then
     hours := floor(full_sec / 3600);
     balance := mod(full_sec,3600);
     full_sec := balance;
     if hours > 1 then
       ret_val := ret_val || to_char(hours) ||' hrs, ';
     else
       ret_val := ret_val || to_char(hours) ||' hr, ';
     end if;
   end if;
   if full_sec > 59 then
     minutes := floor(full_sec / 60);
     balance := mod(full_sec,60);
     full_sec := balance;
     if minutes > 1 then
       ret_val := ret_val||to_char(minutes)||' mins, ';
     else
       ret_val := ret_val||to_char(minutes)||' min, ';
     end if;
   end if;
   ret_val := ret_val||to_char(full_sec)||' secs';
 else
   -- Calculate the time difference in hours,
   -- to three decimal places
   ret_val := to_char((24 * days) + round((full_sec / 3600),3));
 end if;
 return ret_val;
end;
/
grant execute on time_between to public;
create public synonym time_between for time_between;
```

```
select Name, time_between(dob, SysDate, 'N') AS Age
from person
where name='Smith';

NAME            AGE
------------    ------------------------------------------------
Smith            11866 days, 10 hrs, 38 mins, 14 secs
```

Oracle Editing SQL Buffer

Command	**abbrev.**	**Oper. on crnt. line/all lines**
append txt	a text	adds text at the end of a line
change /old/new/	c /old/new/	change old to new in a line
change /txt	c /txt	delete text from a line
clear buffer	cl buff	delete all lines in the buffer
del		delete a line
get file		load file into buffer
input	i	add one or more lines
input txt	iI txt	add text as a line
list	l	list all lines of buffer
list n	l n (n)	list line n and make it current
list *	l *	list crnt. Line
list last	l last	list last line
list m n	l m n	list lines m – n
save file	sav file	save buffer to file

```
SQL> desc user_catalog;
Name            Null?     Type
----------- --------- --------------
 TABLE_NAME  NOT NULL VARCHAR2(30)
 TABLE_TYPE           VARCHAR2(11)

SQL> select * from cat;
TABLE_NAME                TABLE_TYPE
----------------- -----------
ASSIGNED_TO                 TABLE
BONUS                       TABLE
COURSE                      TABLE
CRS_SECTION                 TABLE
DEPT                        TABLE
DEPTMAJOR                   TABLE
DISTANCE                    TABLE
DO_PROJECT                  VIEW
DO_PROJ_SUP                 VIEW
DUMMY                       TABLE
EMAIL_INFO                  TABLE
```

Note: cat is a synonym for user_catalog

```
SQL> desc assigned_to;
 Name                            Null?    Type
 ------------------------------- -------- ----------
 PROJ#                                    NUMBER(4)
 EMP#                                     NUMBER(4)
 ......... .

SQL> select table_name from user_tables;

TABLE_NAME
------------------------------
ASSIGNED_TO
BONUS
COURSE
......... .

SQL> select TABLESPACE_NAME from user_tables;
TABLESPACE_NAME
------------------------------
SYSTEM
SYSTEM
TUTOR
TUTOR  ......... .
```

```
SQL> desc user_tables;
 Name              Null?    Type
 ----------------- -------- --------------
 TABLE_NAME        NOT NULL VARCHAR2(30)
 TABLESPACE_NAME            VARCHAR2(30)
 CLUSTER_NAME               VARCHAR2(30)
 IOT_NAME                   VARCHAR2(30)
 ........ etc.
SQL>  desc user_tab_columns;
 Name                 Null?     Type
 ------------------- -------- -------------
 TABLE_NAME          NOT NULL VARCHAR2(30)
 COLUMN_NAME         NOT NULL VARCHAR2(30)
 DATA_TYPE                    VARCHAR2(106)
 DATA_TYPE_MOD                VARCHAR2(3)
 DATA_TYPE_OWNER              VARCHAR2(30)
 ........ etc.
```

```
SQL> desc user_views;
 Name                           Null?     Type
 ------------------- --------  ------------
 VIEW_NAME                    NOT NULL   VARCHAR2(30)
 TEXT_LENGTH                             NUMBER
 TEXT                                    LONG
 TYPE_TEXT_LENGTH                        NUMBER
 TYPE_TEXT                               VARCHAR2(4000)
 OID_TEXT_LENGTH                         NUMBER
 OID_TEXT                                VARCHAR2(4000)
 VIEW_TYPE_OWNER                         VARCHAR2(30)
 VIEW_TYPE                               VARCHAR2(30)
 SUPERVIEW_NAME


SQL> select view_name from user_views;
VIEW_NAME
--------------------
DO_PROJECT
DO_PROJ_SUP
TEMP1
```

```
SQL> desc user_triggers;
 Name                           Null?    Type
 ------------------------------ -------- ----------------
 TRIGGER_NAME                            VARCHAR2(30)
 TRIGGER_TYPE                            VARCHAR2(16)
 TRIGGERING_EVENT                        VARCHAR2(227)
 TABLE_OWNER                             VARCHAR2(30)
 BASE_OBJECT_TYPE                        VARCHAR2(16)
 TABLE_NAME                              VARCHAR2(30)
 COLUMN_NAME                             VARCHAR2(4000)
 REFERENCING_NAMES                       VARCHAR2(128)
 WHEN_CLAUSE                             VARCHAR2(4000)
 STATUS                                  VARCHAR2(8)
 DESCRIPTION                             VARCHAR2(4000)
 ACTION_TYPE                             VARCHAR2(11)
 TRIGGER_BODY                            LONG
```

```
SQL> select  TRIGGER_NAME from user_triggers;
TRIGGER_NAME
------------------------------
EMP_SAL_RAISE
PERSON1_ST
```

```
select TRIGGER_NAME,TRIGGER_TYPE,TRIGGERING_EVENT,TABLE_OWNER
from user_triggers
where TRIGGER_NAME='PERSON1_ST';

TRIGGER_NAME  TRIGGER_TYPE      TRIGGERING_EVENT TABLE_OWNER
------------  --------------    ---------------- -----------
PERSON1_ST    AFTER EACH ROW    INSERT            SCOTT
```

```
SQL> SET PAGESIZE 66
SQL> COLUMN object_type FORMAT A20
SQL> COLUMN object_name FORMAT A30
SQL> COLUMN status FORMAT A10
SQL> BREAK ON object_type SKIP 1
SQL> SELECT   object_type, object_name, status
   FROM user_objects
   WHERE object_type IN ('PACKAGE','PACKAGE BODY',
                'FUNCTION','PROCEDURE',
                'TYPE','TYPE BODY',
                       'TRIGGER');

OBJECT_TYPE     OBJECT_NAME        STATUS
--------------- ------------------ ----------
FUNCTION        BUSINESS_DAYS      VALID

TRIGGER         PERSON1_ST         VALID
```

```
select text
from user_source
where name='WORKING_DAYS';

TEXT
---------------------------------------------------------------------------------------
FUNCTION working_days(date1 IN DATE, date2 IN DATE)
RETURN NUMBER IS workdays NUMBER;
BEGIN
 workdays := TRUNC(date2) - TRUNC(date1) + 1
        - ((TRUNC(to_date(date2,'D'))-TRUNC(to_date(date1),'D'))/7)*2;
  IF TO_CHAR(date2,'D') = '7' THEN
    workdays := workdays - 1;
  END IF;
  IF TO_CHAR(date1,'D') = '1' THEN
    workdays := workdays - 1;
  END IF;
  RETURN(workdays);
end;
```

```
select text
from user_source
where name=PERSONV_ST';

TEXT
--------------------------------------------------------------------------------
TRIGGER personv_st
INSTEAD OF INSERT ON PERSONV
FOR EACH ROW
DECLARE STATUS CHAR(4);
BEGIN
STATUS := 'Poor';
IF (:new.income > 20000) THEN
STATUS := 'Med'; END IF;
IF (:new.income > 60000)THEN
STATUS := 'Rich'; END IF;
INSERT INTO person VALUES(:new.name, :new.dob, :new.income, STATUS);
END person_st;
```

```
select object_name
from user_procedures;

OBJECT_NAME
-------------------------------
WORKING_DAYS
```

```
select text
from user_source
where name='WORKING_DAYS';
```

```
TEXT
------------------------------------------------------------------------------
FUNCTION working_days(date1 IN DATE, date2 IN DATE)
RETURN NUMBER IS workdays NUMBER;
BEGIN
 workdays := TRUNC(date2) - TRUNC(date1) + 1
        -
((TRUNC(to_date(date2,'D'))-TRUNC(to_date(date1),'D'))/7)*2;
  IF TO_CHAR(date2,'D') = '7' THEN
    workdays := workdays - 1;
  END IF;
  IF TO_CHAR(date1,'D') = '1' THEN
    workdays := workdays - 1;
  END IF;
  RETURN(workdays);
end;
```

Course Notes
Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

This document in electronic form, bearing a CopyForward permission, could be used for personal use and/or study, free of charge. Anyone could use it to derive updated versions. The derived version must be published under CopyForward. All authors of the version used to derive the new version must be included in the updated version in the existing order, followed by name(s) of author(s) producing the derived work

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

Relation Algebra, Bags and Constraints

Bipin C. DESAI

BAGS

In a set, there are no duplicates.

A set (collection of similar objects) having multiple occurrences of one or more members is called a “bag”.

Implementation of relational model allow a relation(table) to have duplicates.

This is specially so for intermediate results(a convenience) and if no primary keys are defined for a table.

Thus in a relation that is a bag, duplicate tuples are allowed(though not required – so a bag may have no duplicates at a given point in time.)

A stored table is not a bag; since constraints enforcements require that duplicates are not stored **(each row has an UNIQUE row identifier)**

Instance R which is not a bag:

A	B	C
a1	b1	c1
a1	b2	c2
a3	b1	c1
a4	b2	c2
a5	b5	c5

Π_{BC}R without eliminating the duplicates tuples is a bag!

B	C
b1	c1
b2	c2
b1	c1
b2	c2
b5	c5

Faster projection operations: no need to check duplicates for each tuple in the output relation

Correct computation compute the *average* of values under attribute B in the projection of R.

Faster bag Unions: Computing (**R** $\cup_{\mathbf{Bag}}$ **S**) is cheaper than (**R** $\cup_{\mathbf{Set}}$ **S**).

Given R has *n* tuples and S has *m* tuples, then the costs of evaluating these queries would be **O**$(n + m)$ and **O**$(n * m)$, respectively.

R $\cup_{\mathbf{Bag}}$ **S** ≡ **R** $\cup_{\mathbf{B}}$ **S** **R** $\cup_{\mathbf{Set}}$ **S** ≡ **R** $\cup_{\mathbf{S}}$ **S** ≡ **R** ∪ **S**

R $\cup_B$ S (**bag union** of R and S): the bag of tuples that are in R, in S**,** or in both. If a tuple *t* appears *n* times in R, and *m* times in **s**, then *t* appears *n+m* times in bag R $\cup_B$ S

R $\cup_B$ S = $\{ t{:}k \mid t{:}n \in R \wedge t{:}m \in S \wedge k = n+m \}$

R $\cap_B$ S(**bag intersection** of R and S): the bag of tuples that appear in both R and S**.** If a tuple *t* appears *n* times in R, and *m* times in S, then the number of occurrences of *t* in bag R $\cap_B$ **s** is *min(n,m)*

R $\cap_B$ S = $\{ t{:}k \mid t{:}n \in R \wedge t{:}m \in S \wedge k = min(n,m) \}$

R $-_B$ S (**bag difference** of R and S): is defined as follows:

R $-_B$ S = $\{ t{:}k \mid t{:}n \in R \wedge t{:}m \in S \wedge k = max(0, n-m) \}$

S $-_B$ R = $\{t{:}k \mid t{:}n \in R \wedge t{:}m \in S \wedge k = max(0, m-n) \}$

R

A	B
a1	b1
a2	b2
a1	b1
a1	b1

S

A	B
a1	b1
a2	b2
a2	b2
a3	b3

R

A	B
a1	b1
a2	b2
a1	b1
a1	b1

S

A	B
a1	b1
a2	b2
a2	b2
a3	b3
a1	b1

If there was another (a1,b1)

$R \cup_B S$

A	B
a1	b1
a2	b2
a1	b1
a1	b1
a1	b1
a2	b2
a2	b2
a3	b3

$R \cap_B S$

A	B
a1	b1
a2	b2
a1	b1

min(1,3)

min(1,2)

min(2,3)

$R -_B S$

A	B
a1	b1
a1	b1

max(0,3-1)

$S -_B R$

A	B
a2	b2
a3	b3

BAG PROJECTION

Let **R** be a relation scheme, and R be a bag over **R**.

The **projection** operator is used to produce, from R**,** a new bag that has only some of **R**’s columns

If elimination of one or more attributes during the projection causes the same tuple to be created from several tuples, these **multiple tuples are not eliminated** from the result of a bag-projection

A	B	C
a1	**b1**	**c1**
a1	**b2**	**c2**
a3	**b1**	**c1**
a4	**b2**	**c2**
a5	**b5**	**c5**

B	C
b1	c1
b2	c2
b1	c1
b2	c2
b5	c5

BAG SELECTION $\sigma_C R$

The tuples in the output relation are those that satisfy the predicate **C**, which involves attributes of R

Duplicates are eliminated in a set selection but **not so** from the result of a bag-selection

Note: The selection operation σ in RA is not the same as the **SELECT** clause in SQL which is the projection part of the DML component of SQL

A	B	C
a1	b1	c1
a1	b1	c2
a3	b1	c1
a4	b2	c2
a5	b5	c5

R - not a bag

A	B	C
a1	b1	c1
a1	b1	c2
a3	b1	c1

$\sigma_{B = b1}$ (R):

Not a bag

A	B	C
a1	b1	c1
a1	b1	c2
a1	b1	c1
a1	b2	c2
a5	b5	c5

S - a bag

A	B	C
a1	b1	c1
a1	b1	c2
a1	b1	c1

$\sigma_{B = b1}$ (S):

A bag

Cartesian Product of Bags

Given R and S, then R $\times_B$ S is the bag of tuples formed by concatenating pairs of tuples, the first of which comes from R and the second from S.

R $\times_B$ S $= \{ t_1.t_2 \mid t_1 \in R \wedge t_2 \in S \}$

As in the set cartesion product, each tuple of R is paired with each tuple of S: however, in bag product each tuple is used regardless of whether it is a duplicate or not.

Hence, if a tuple t_1 appears ***m*** times in a relation R, and a tuple t_2 appears ***n*** times in relation S, then tuple $t_1.t_2$ appears ***m*n*** times in the bag-product R $\times_B$ S

Joins of Bags $\bowtie_{B(predicate)}$

The join of bags R $\bowtie_{B(predicate)}$ S is computed in the same way as the join of sets; however, duplicates are not eliminated!

A	B
a1	b1
a1	b1

R

B	C
b2	c2
b3	c3
b3	c3

S

A	B
a1	b1
a1	b1

R

B	C
b1	c2
b3	c3

T

A	R.B	S.B	C
a1	b1	b2	c2
a1	b1	b3	c3
a1	b1	b3	c3
a1	b1	b2	c2
a1	b1	b3	c3
a1	b1	b3	c3

$R \times_B S$

A	B	C
a1	b1	c2
a1	b1	c2

$R \bowtie_{B(R.B=S.B)} T$

Constraints on Relations

Relational algebra offers a convenient way to express a wide variety of constraints, such as referential integrity and FD's.

There are **two ways** to express constraints in RA

If *r* is a relational algebraic expression, then $r = \Phi$ is a constraint that says **"the value of r must be empty"**

If *r* and *s* are relational algebraic expressions, then $r \subseteq s$ is a constraint that says *"every tuple in the result of r is in the result of s"* **(even when *r* and *s* are bags)**

A RA constraint may be expressed in more than one way; i.e.

$\mathbf{r} \subseteq \mathbf{s}$ could be written as $\mathbf{r - s} = \Phi$

If $\mathbf{r} \subseteq \mathbf{s}$, $\therefore$ no tuple in **r** that is not in **s**, and hence $\mathbf{r - s} = \Phi$

The constraint $\mathbf{r} = \Phi$ could be rewritten as: $\mathbf{r} \subseteq \Phi$

Referential integrity

If we have a value ***v*** in a tuple *t* of a relation R, then ***v*** must also appears as a component of some tuple *s* of relation S

Example: if we have a tuple *(s,c,g)* in relation **Enrol***(sid,crsno,grade),* then there must be a student with *sid = s* and a course with *crsno = c* such that *s* has taken/taking *c*.

*The values s and c in **Enrol** are "referring" to some values outside this relation, and these values **must** exist in the **Student** and **Course** relations* **Course**(*crsno, name, credits*)

$$\pi_{crsno}\ \textbf{Enrol} \subseteq \pi_{crsno}\ \textbf{Course}$$

or equivalently

$$\pi_{crsno}\ \textbf{Enrol}\ - \pi_{crsno}\ \textbf{Course} = \Phi$$

Functional Dependency

Definition: If two tuples of a relation R agree on the attributes X, then they must also agree on the attributes Y.

Student*(sid,name,dob, gender), sid* → *name*

To express the FD: *sid* → *name* in RA, construct **pairs of Student** tuples, using **Cartesian product**, and see if there is a violation of this FD, using selection with *sid*s equal but *name*s not.

To **assert the constraint,** we equate the result must be null.

$\rho(S, \rho(S1, \textbf{Student}) \times \rho(S2, \textbf{Student}))$

$\boldsymbol{\sigma}_{S1.sid=S2.sid \,\hat{}\, S1.name \neq S2.name}\ S = \boldsymbol{\Phi}$

Domain Constraints

Empl (*<u>Empl#</u>, name,dob, gender, salary*)

- To express the domain constraint:

The only valid values for the attribute **gender** are **'F'** and **'M'**

$$\sigma_{\text{gender} \neq \text{'F' AND gender} \neq \text{'M'}}(\textbf{Empl}) = \Phi$$

- To express the following constraint?

Maximum salary of every employee is $30,000

$$\sigma_{\textit{salary} > 30000}(\textbf{Empl}) = \Phi$$

These are examples of ***domain constraints***

Bags in DBMS

```
create table dept(
 dcode number(3),
 dname varchar2(30),
 location varchar2(30))
/

insert into dept values (100, 'CS', 'EV300');
insert into dept values (100, 'CS', 'EV300');
insert into dept values (100, 'CS', 'EV300');
/
```

```
SQL> select * from dept;
    DCODE DNAME                          LOCATION

---------- ------------------------------ ------------------------------

      100      CS                        EV300
      100      CS                        EV300
      100      CS                        EV300
      100      CS                        EV300
```

Without a primary key ORACLE/MySQL/MariaDB ALLOWS DUPLICATES!

```
SQL> desc dept;
 Name                               Null?   Type

 ---------------------------------- -------- ----------------------------

 DCODE                                        NUMBER(3)
 DNAME                                        VARCHAR2(30)
 LOCATION                                     VARCHAR2(30)
```

```
SQL> alter table dept add(constraint pk_const primary key(dcode) );
ERROR at line 1:
ORA-02437: cannot validate (SCOTT.PK_CONST) –
        primary key violated
```

Remove duplicate records

```
delete  from dept
where   rowid in (
      select rowid
      from   dept
      minus
      select max(rowid)
      from   dept  d group by d.dcode);
3 rows deleted.
```

```
SQL> select * from dept;

DCODE   DNAME        LOCATION
----------  --------------  ------------------------
     100      CS                 EV300
```

```
SQL> alter table dept add(constraint pk_const primary key(dcode) );

Table altered.

SQL> desc dept;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 DCODE                                     NOT NULL NUMBER(3)
 DNAME                                              VARCHAR2(30)
 LOCATION                                           VARCHAR2(30)
```

How about MySql:

```
create table dept(
 dcode numeric(3),
 dname varchar(30),
 location varchar(30));
```

No primary key

```
insert into dept values (100, 'CS', 'EV300');
insert into dept values (100, 'CS', 'EV300');
insert into dept values (100, 'CS', 'EV300');
```

```
mysql> select * from dept;
+-------+-------+----------+
| dcode | dname | location |
+-------+-------+----------+
|   100 | CS    | EV300    |
|   100 | CS    | EV300    |
|   100 | CS    | EV300    |
+-------+-------+----------+
```

MYSQL ALLOWS DUPLICATES

Eliminate Duplicates rows

```
CREATE TEMPORARY TABLE Temp
     select distinct * from dept; --Temp table
delete from dept;
insert into dept
      select * from Temp;

mysql> desc dept;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| dcode    | decimal(3,0) | YES  |     | NULL    |       |
| dname    | varchar(30)  | YES  |     | NULL    |       |
| location | varchar(30)  | YES  |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
3 rows in set (0.00 sec)
```

```
mysql> alter table dept modify column dcode numeric(3) primary
   key;
```

```
mysql> desc dept;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| dcode    | decimal(3,0)  |      | PRI | 0       |       |
| dname    | varchar(30)   | YES  |     | NULL    |       |
| location | varchar(30)   | YES  |     | NULL    |       |
+----------+---------------+------+-----+---------+-------+
```

```
mysql> insert into dept values (100, 'CS', 'EV300');
ERROR 1062: Duplicate entry '100' for key 1
```

```
MariaDB [test]>select * from dept;
+-------+-------+----------+
| dcode | dname | location |
+-------+-------+----------+
|   100 | CS    | EV300    |
+-------+-------+----------+
1 row in set (0.000 sec)
```

Course Notes

Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

Data Models

Past & Future(the past re-dressed)!

Bipin C. DESAI

To be used in the spirit of copy-forward! https://users.encs.concordia.ca/~bcdesai/CopyForward.pdf

Early methods of managing data: did **not** use SQL!

Definition of data was locked in application programs
Data in compact form on external media
punched cards (10x80), punched paper tape, magnetic tapes

Computer Museum

Hierarchical Data Model (HDM)

The HDM introduced in 1960s by IBM in IMS

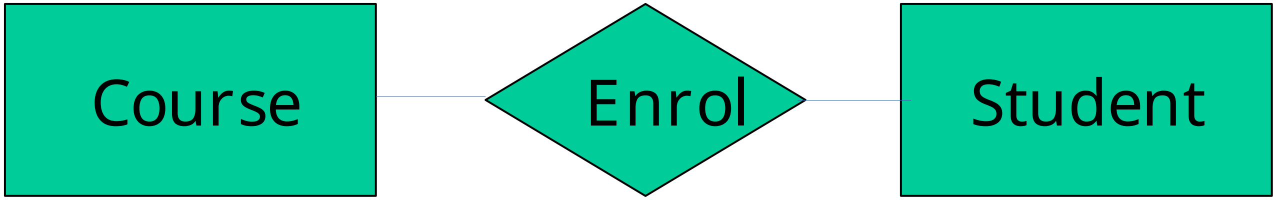
-it is based on the parent-child model.

Parent record type, children record types

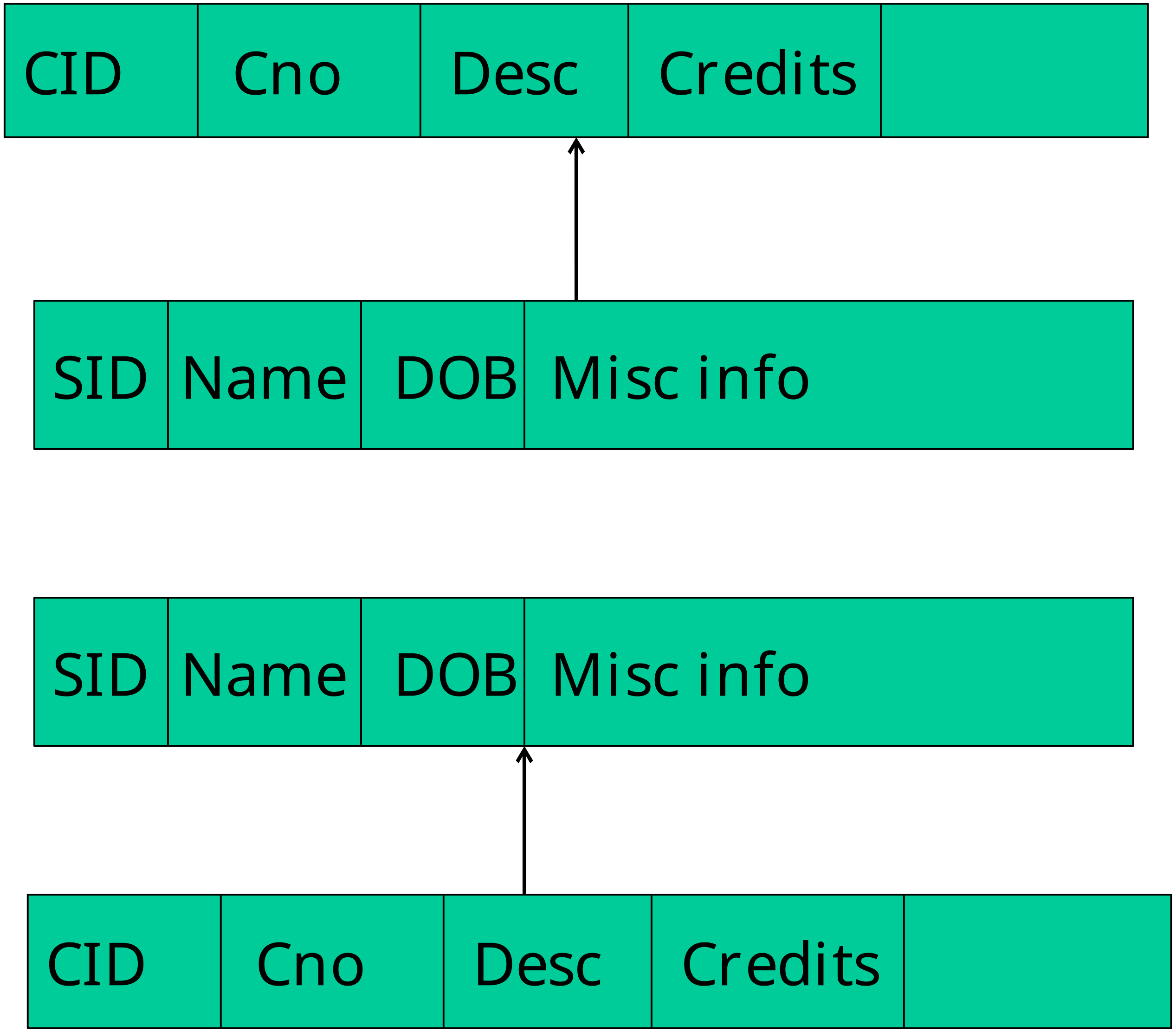
These record types are organized in the form of a rooted tree (hence, no cycles).

Only one-to-many or one-to-one relationship can be represented

Many-to-many relationship requires duplication



Did not use SQL - but not a NoSQL DB!!



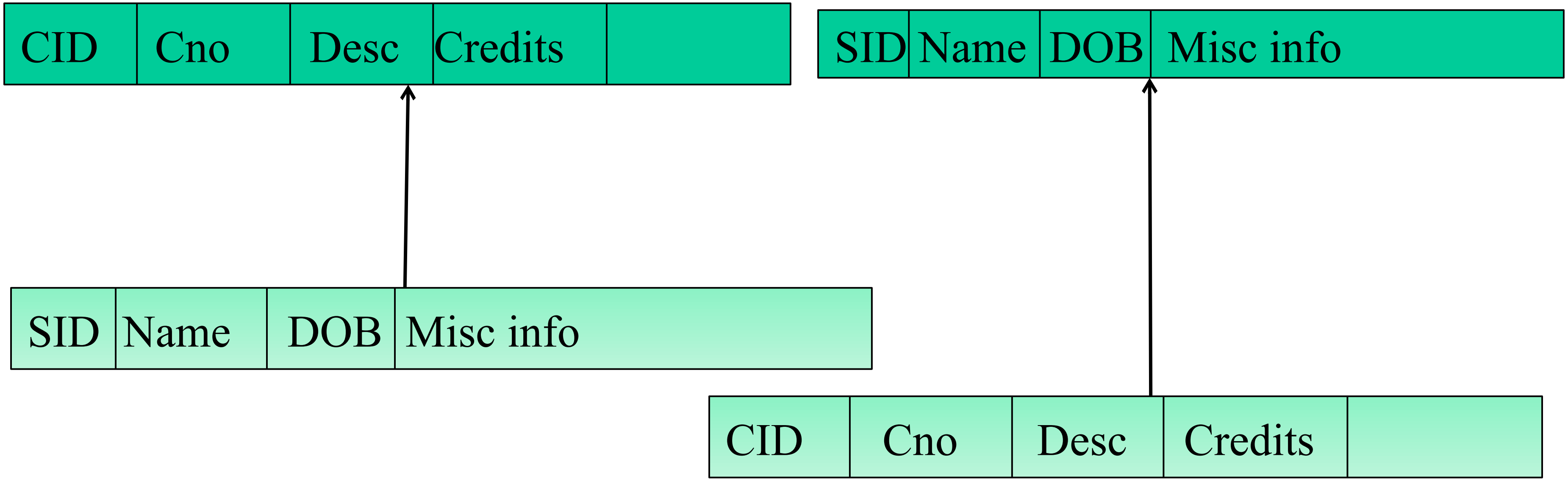
CID | Cno | Desc | Credits

SID | Name | DOB | Misc info

SID | Name | DOB | Misc info

CID | Cno | Desc | Credits

Duplication of records and many-to-many relationship in HDM



CID	Cno	Desc	Credits	

SID	Name	DOB	Misc info

SID	Name	DOB	Misc info

CID	Cno	Desc	Credits	

Virtual records and implementation of many-to-many relationship in HDM

HDM databases has features for ‘fixed pattern” usage:
Pros:
Since the structure is fixed data access faster
HDM the relationships is fixed - easy for data integrity
HDM represent many data in normally used
– all banking transactions belong to a given account
Queries are predictable and uses the hierarchy

Cons:
Duplication for many-to-many relationships;
Difficult to change schema -

IMS currently is in version 15+ and is available for z/OS
offers SQL for IMS!

Network Data model (DBTG)

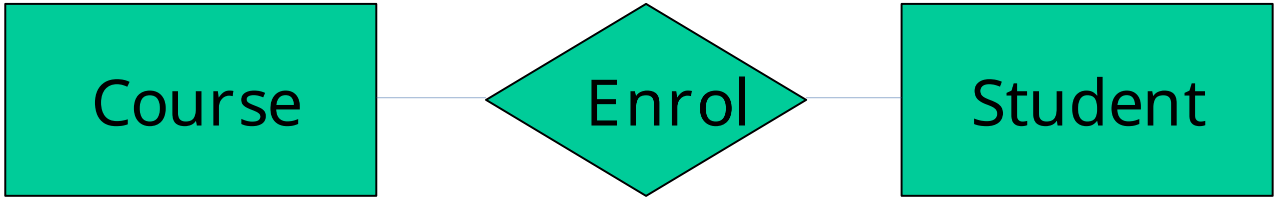
In the NDM is based on the set with an owner and a member record types concept : the DBTG set

Each DBTG set can have any number of *set occurrences* (a number of instances of linked records).
many-to-many links are not allowed,
each set occurrence has precisely one owner,
and has zero or more member records.

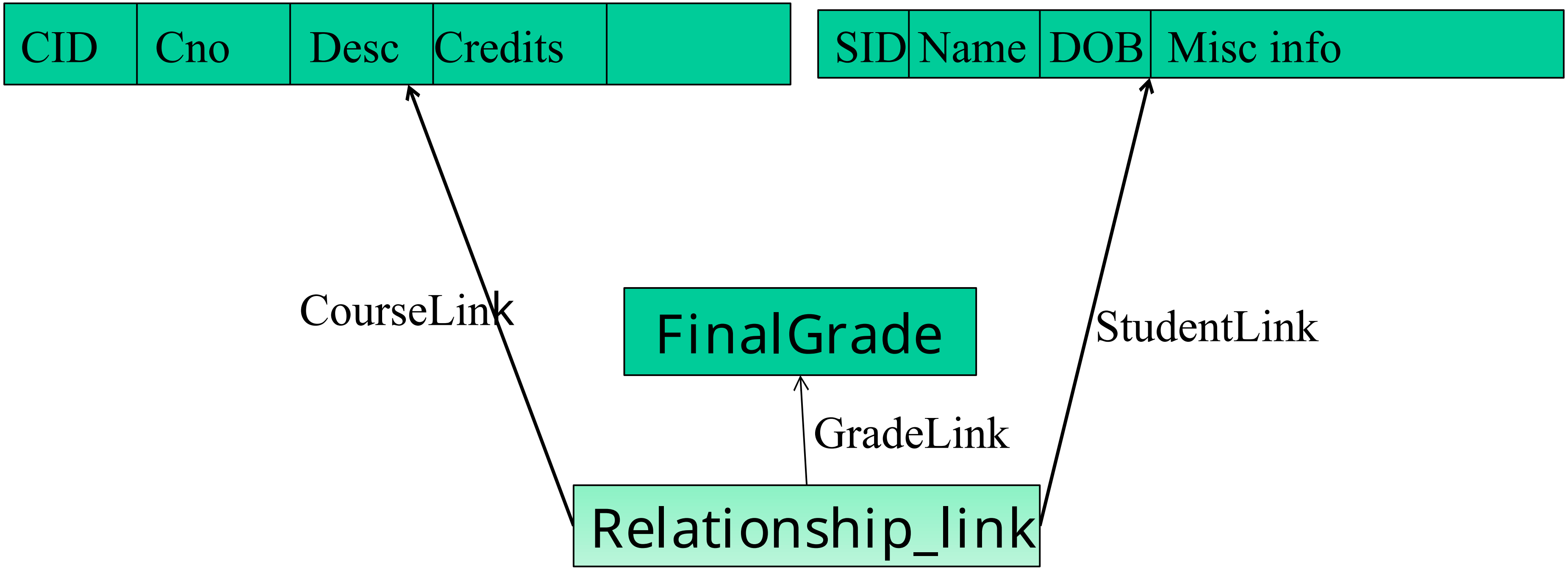
A member record in a set can participate in only one occurrence of a given set at any point.

However, any record can participate simultaneously in several set occurrences of *different* DBTG sets.

Did not use SQL - but not a NoSQL DB!!



Representing a many-to-many relationship in NDM by two sets and a dummy record to store the attribute of the relationship



Object Oriented World & ODL

- In an object oriented design, the "**part of the world**" we want to model is thought of as being **composed of objects**
- Everything is an **object** and **"similar"** objects *are instances of objects called* **class**
    - *People, Employees, Bank accounts, Students, Course, Airline flights*
- A **class** simply represents a grouping of **similar objects**
- Every **object** is an **instance** of a **class** and has a unique object identification (OID)
- All objects that are instances of the same class have the same **properties** and **behavior(***interaction of objects***)**

ODL (Object Definition Language) is a proposed standard language for specifying structure of databases

- **ODL** is an extension of **IDL** (Interface Description Language), a component of **CORBA** (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture)

Class Declarations

- A declaration of a **class** in ODL, consists of:
  - The keyword **class**
  - The **name** of the class
  - A bracketed { …} list of **properties** of the class

```
class<name> {
    <list of properties>
  };
```

```
class student {
    …
  };
```

Properties of ODL classes

- ODL classes can have three kinds of properties:
  - **Attributes**
    - □ properties whose types are built from **primitive/basic types** such as integers, strings,…
  - **Relationships**
    - □ properties whose type is either a **reference** to an object

      (x-one) or a **collection** of such references (x-many),

      where x could be one or many.
  - **Methods**
    - □ **functions** that may be applied to objects of the class

Attributes in ODL

- Attributes are the **simplest kinds** of properties
- An attribute **describes some aspect of an object** by associating, with the object, a value of some simple **type**
- For example, attributes of a **Student** object
  - Student ID
  - Name
  - Address
  - E-mail

Keys in ODL

- In **ODL**, we declare keys using the keyword **key**
  - If a key has more than one attribute, we surround them by (…)
    - □ Example: (two attributes forming a key)

```
 class Movie
          ( extent  Movies key (title, year) ) {
          attribute string title;
          …        };
```

  - If a class has more than one key, we may list them all, separated by commas
    - □ Example: (A class with two keys)

```
 class Employee
           (extent  Employees key empID, SIN) {…};
```

Single-Value Constraints in ODL

- Often, we should *enforce* properties in the database saying that there is **at most one** value playing a particular role
  - For example**:**
    - that a movie object has a **unique** title, year, length, and film type
    - that a movie is owned by a **unique** studio

Single-Value Constraints

- In **ODL:**
  - An attributes is not of collection type
    (Set, Bag, Array, … are example of **collection types**.)
  - A relationship is either a class type or (a single use of) a collection type constructor applied to a class type.
- Recall that in **E/R:**
  - attributes are **atomic**
  - an arrow (→) or a value on the connecting line can be used to express the type of relationship(multiplicity)
  - How about multi-valued? attributes (No) but relationships (Yes)

Type system

A **type system** consists of

- **Basic types**
- **Type constructors:** recursive rules whereby **complex types** are built from simpler ones

- **Atomic types**

  Integer

  Char

  Boolean

  Float

  Character String

  Enumeration

  Enumeration is a **list of names** declared to be **synonyms for integers**

- **Class types**
  - Movie

Type constructors in ODL

- **Set**
  - Set <integer>
  - Set <Movie>
- **Bag**
  - Bag <integer>
  - Bag <Movie>
- **Array**
  - Array <integer, 10>
  - Array <Movie, 3>
- **Structure**
  - Struct Address {string street, string city}
- **List**
  - List <integer>
  - List <Student>
- **Dictionary** <keyType, valueType>
  - Dictionary<Student, string>

- **Note:**
  - Set, Bag, Array, List and Dictionary are called **collection types**
  - Collection type cannot be applied repeatedly (nested)
    - E.g., it is **illegal** to write Set<Array<integer>>

Example

```
class Movie {
  attribute string title;
  attribute integer year;
  attribute integer length;
  attribute enum Film {Colour, BlackAndWhite} filmType; };
```

("The Barbarian Invasions ", 2003, 112, Colour)
is an object, i.e., an instance of the class Movie

```
 class Star {
   attribute string name;
   attribute Struct Address {
         string street,
         Array <char, 10> city
                  } homeAddress;
     attribute Struct Address officeAddress; };
```

(structure with non-atomic type)

More Examples

```
class Student {
    attribute string ID;
    attribute string lastName;
    attribute string firstName;
    attribute integer dob;
    attribute string program;
    attribute Struct Address {
          string street,
          string city
               } homeAddress;
    };
```

```
class Course {
    attribute string courseNumber;
          attribute string courseName;
          attribute integer NoOfCredits;
    attribute string department;
          };
```

Expressing Relationships in ODL

- How are **Movies** and **Stars** related?
- **Movies** have actors/actresses(**Stars)**, and **Stars** have roles in **Movies!**
- Every movie has a star (or stars)
- In ODL the interaction of classes is expressed by a construct called "relationship"!
- To take into account the fact that a relationship could involve more than one instance of an object from the related class it is expressed as a Set
- Note: In ODL relationship(s) is(are) stored in an object as "OID pointer(s)"; such relationship(s) is(are) not attribute(s)!

Relationship in ODL: an Example

- **starOf** is a relationship between **Movie** and **Star**

```
class Movie {
        attribute string title;
        attribute integer year;
        attribute integer length;
        attribute enum Film {Colour, B&W} filmType;
        relationship Star starOf;
        };
```

- How are **Movies** and **Stars** related?
- Not only every movie has a star
- But also every star has acted in some movie
- To fix this in the **Star** class, we should add the line:

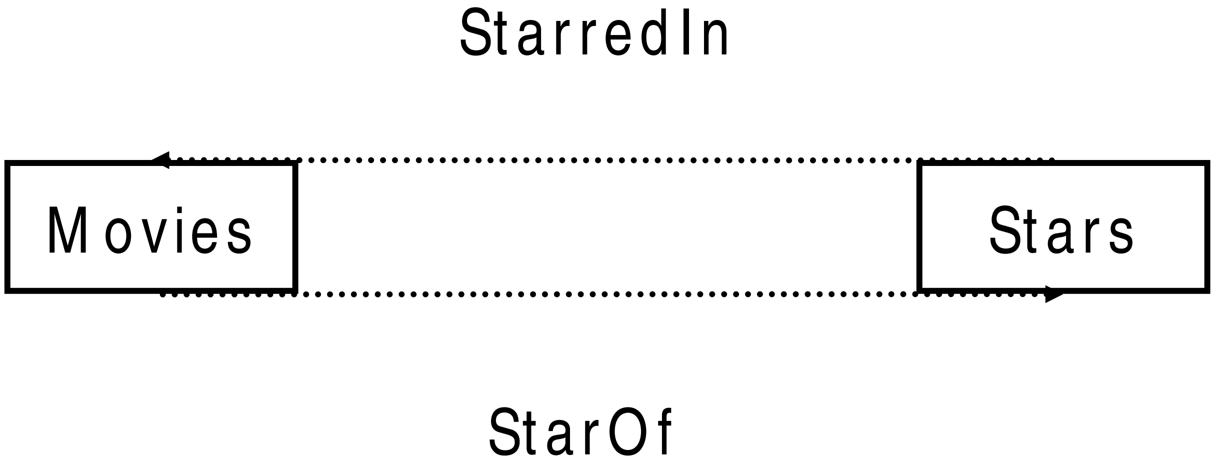
  relationship Movie starredIn;

```
class Star {
     attribute string name;
     attribute Struct Address {
               string street,
               string city
               } address;
     relationship Movie starredIn;        };
```

Is there a problem here?
Hint: inverse relationship

Inverse Relationships

- We are omitting a very important aspect of the relationship between movies and stars
- We need a way to ensure that if a star **S** is ***starOf*** movie **M**, then movie **M** is ***starredIn*** for star **S**
- In ODL that is done by defining **inverse** of a relationship for each class.



```
class Movie {
    attribute string title;
    attribute integer year;
    attribute integer length;
    attribute enum Film {colour, B&W} filmType;
    relationship Star starOf
                inverse Star::starredIn;
};
```

What is the problem here!

- how many actors in a movie?
- how many movies credits for an actor?

```
class Star {
    attribute string name;
    attribute Struct Address {
          string street,
           string city
    } address;
    relationship Movie starredIn
       inverse Movie::starOf;
};
```

Relationships in ODL

- Our model is not quite complete: it is missing an important point!
- A movie typically has several actors and each actor is featured in many movies.
- To fix this, we need to express the relationship as a set:

relationship Set<Star> stars;

```
class Movie {
   attribute string title;
   attribute integer year;
   attribute integer length;
   attribute enum Film {colour, B&W} filmType;
        relationship Set<Star> starOf
               inverse Star::starredIn;
  };
```

Why is this not a set?

```
class Star {
   attribute string name;
   attribute Struct Address {
        string street,
        string city
   } address;
   relationship Set<Movie> starredIn
     inverse Movie::starOf;
};
```

The inverse relationship only specifies the name of the relationship in Star;
the set is in Star not in Movie

What about attributes of a relationship?

- Suppose we introduce another class, **Studio**, representing companies that produce movies
- How are **Movies** and **Studios** related? Every **Studio** owns several **Movies**

```
class Studio {
        attribute string name;
        attribute string address;
        relationship Set<Movie> owns  inverse Movie::ownedBy;
        };
```

- What about inverse? Every **Movie** is owned by some **Studio**

```
class Movie {
        attribute string title;
        attribute integer year;
        attribute integer length;
        attribute enum Film {color, blackAndWhite} filmType;
        relationship Set<Star> starOf inverse Star::starredIn;
        relationship Studio ownedBy  inverse Studio::owns;
        };
```

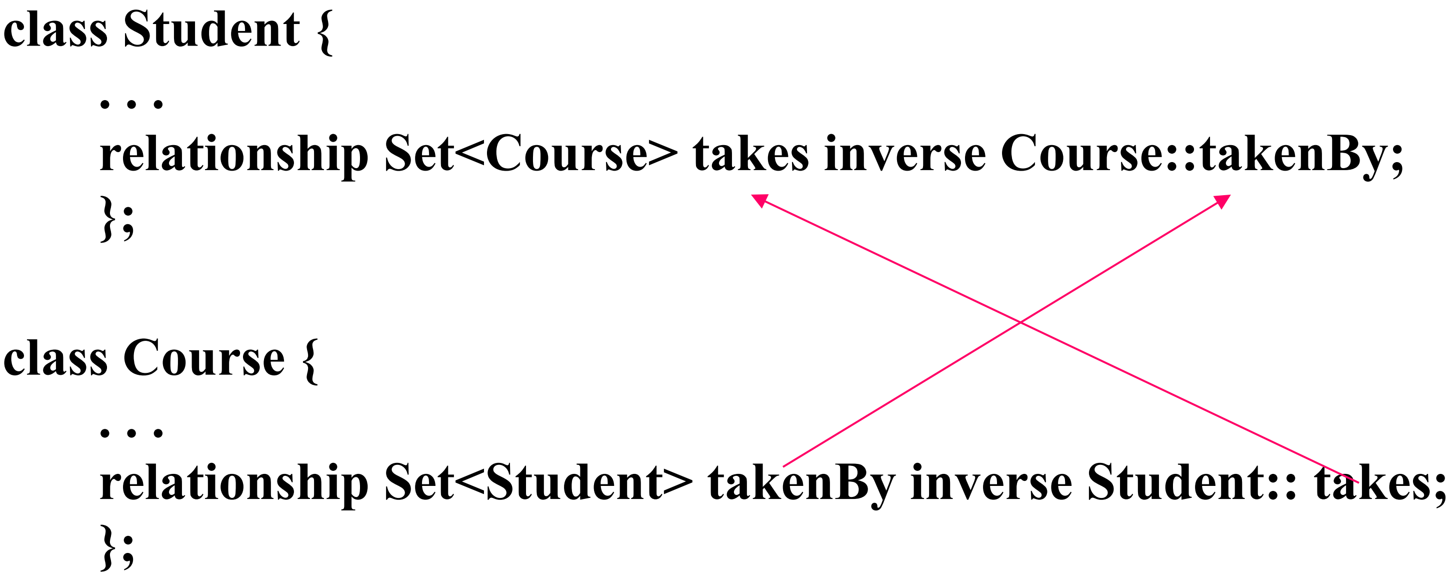
Multiplicity of relationships

- **In general**, when we have a pair of inverse relationships, there are **four** cases:
  - The relationship is unique in both directions (one case)
  - The relationship is unique in just one direction (two cases)
  - The relationship is not unique in any direction (one case)
  - The *multiplicity* thus refers to the one of these relationships; also denoted as 1-1 (one-one), 1-M (one-many), M-1 (many-one), and M-N (many-many).

Multiplicity of relationships: many-many

- A **many-many** relationship from a class **C** to a class **D** is one in which, for each C there is a set of **D**s associated with **C;** in the inverse relationship, a set of **C**s is associated with each D

Example: *each student can take many courses and each course can be taken by more than one student*



```
class Student {
    . . .
    relationship Set<Course> takes inverse Course::takenBy;
    };

class Course {
    . . .
    relationship Set<Student> takenBy inverse Student:: takes;
    };
```

Multiplicity of relationships: many-one

- A **many-one** relationship from class **C** to a class **D**, is one where for each **C** there is a at most one **D**, but no such a constraint in the reverse direction (similarly for one-many

Example, many employees may work in the same department, but each employee works only in one department

```
class Department {
    . . .
    relationship Set< Employee > workers inverse
       Employee::worksIn;
    };

class Employee {
    . . .
    relationship Department worksIn inverse
       Department::workers;
    };
```

Note: There is one-to-many relationship from Employee to Department

Multiplicity of relationships: one-one

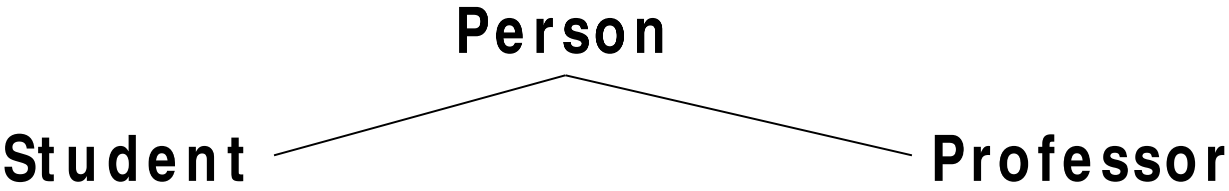
- A **one-one** relationship from class **C** to class **D** is one that for each **C** there is a at most one **D**, and conversely, for each **D** there is at most one **C**

Example: each department has at most one employee as its manager and each employee can manage at most one department

```
class Employee {
    . . .
    relationship Department ManagerOf
      inverse  Department::manager;
    };
class Department {
    . . .
    relationship Employee manager
     inverse Professor:: ManagerOf;
    };
```

Inheritance in Object Oriented System and Subclasses

- Objects can be organized into a hierarchical inheritance structure
- A child class (or subclass) will inherit properties form a parent class (or all superclasses) higher in the hierarchy.
- Often, a class contains objects that have **special properties** not associated with all members of the class
- If so, we find it useful to organize the class into ***subclasses***, each subclass having its **own special** attributes and/or relationships



Person

Student

Professor

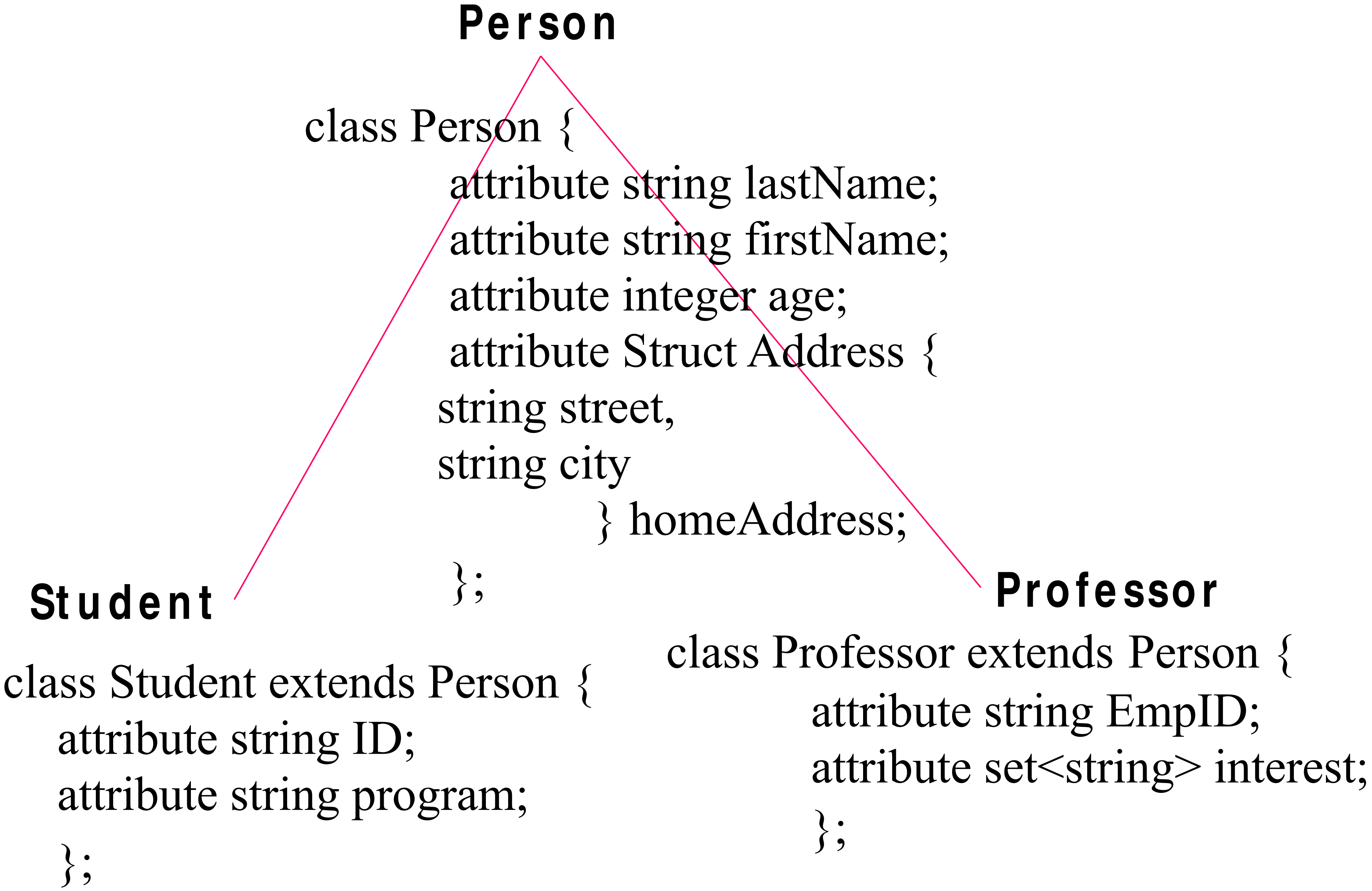
Subclasses in ODL

- We define a class ***C*** to be a subclass of another class ***D*** by following the name ***C*** in its declaration with a keyword **extends** and the name ***D***

```
class Cartoon extends Movie {
    relationship Set<Star> voices;
    };
```

A subclass *inherits* all the properties of its superclasses

So, each cartoon object has *title, year, length, filmType*, and inherits relationships *stars* and *ownedBy* from Movie, in addition to its own relationship *voices*.



Person

```
class Person {
        attribute string lastName;
        attribute string firstName;
        attribute integer age;
        attribute Struct Address {
       string street,
       string city
                } homeAddress;
        };
```

Student

```
class Student extends Person {
   attribute string ID;
   attribute string program;
   };
```

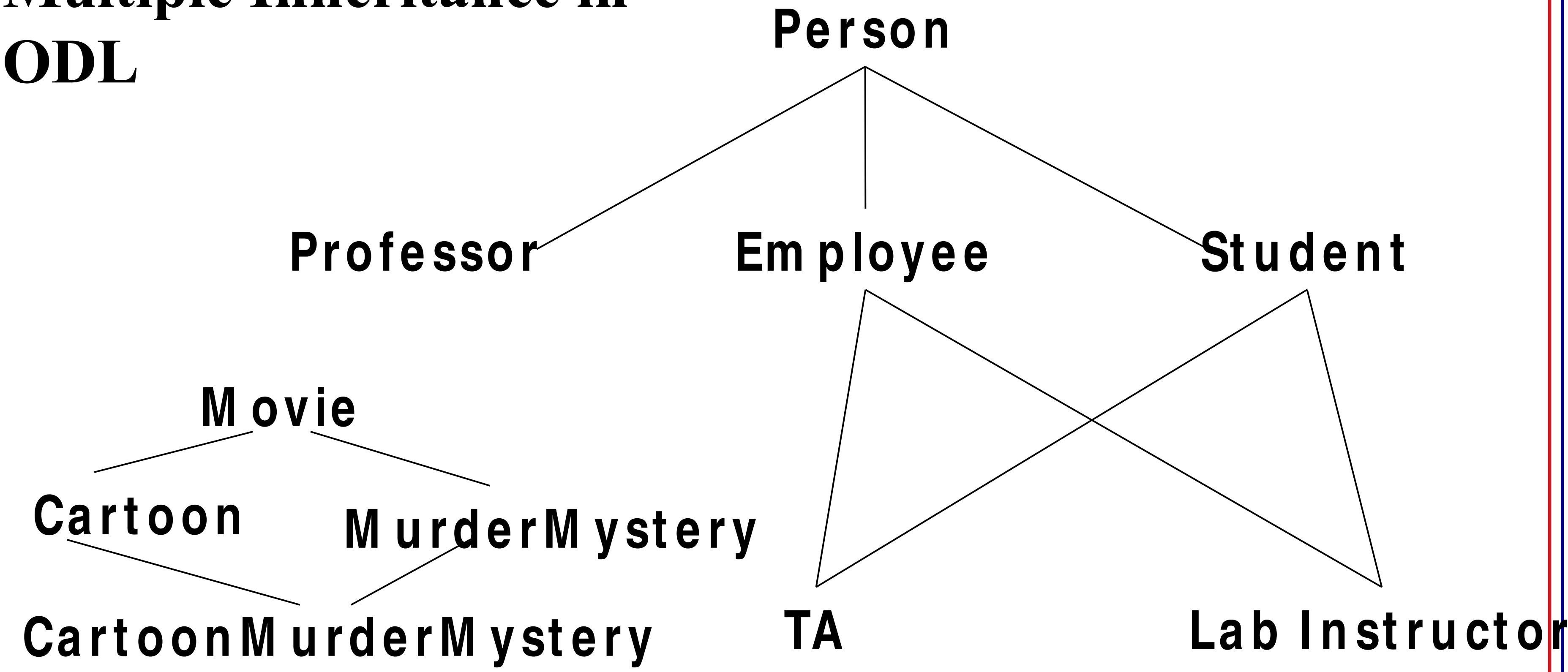
Professor

```
class Professor extends Person {
        attribute string EmpID;
        attribute set<string> interest;
        };
```

Inheritance in ODL

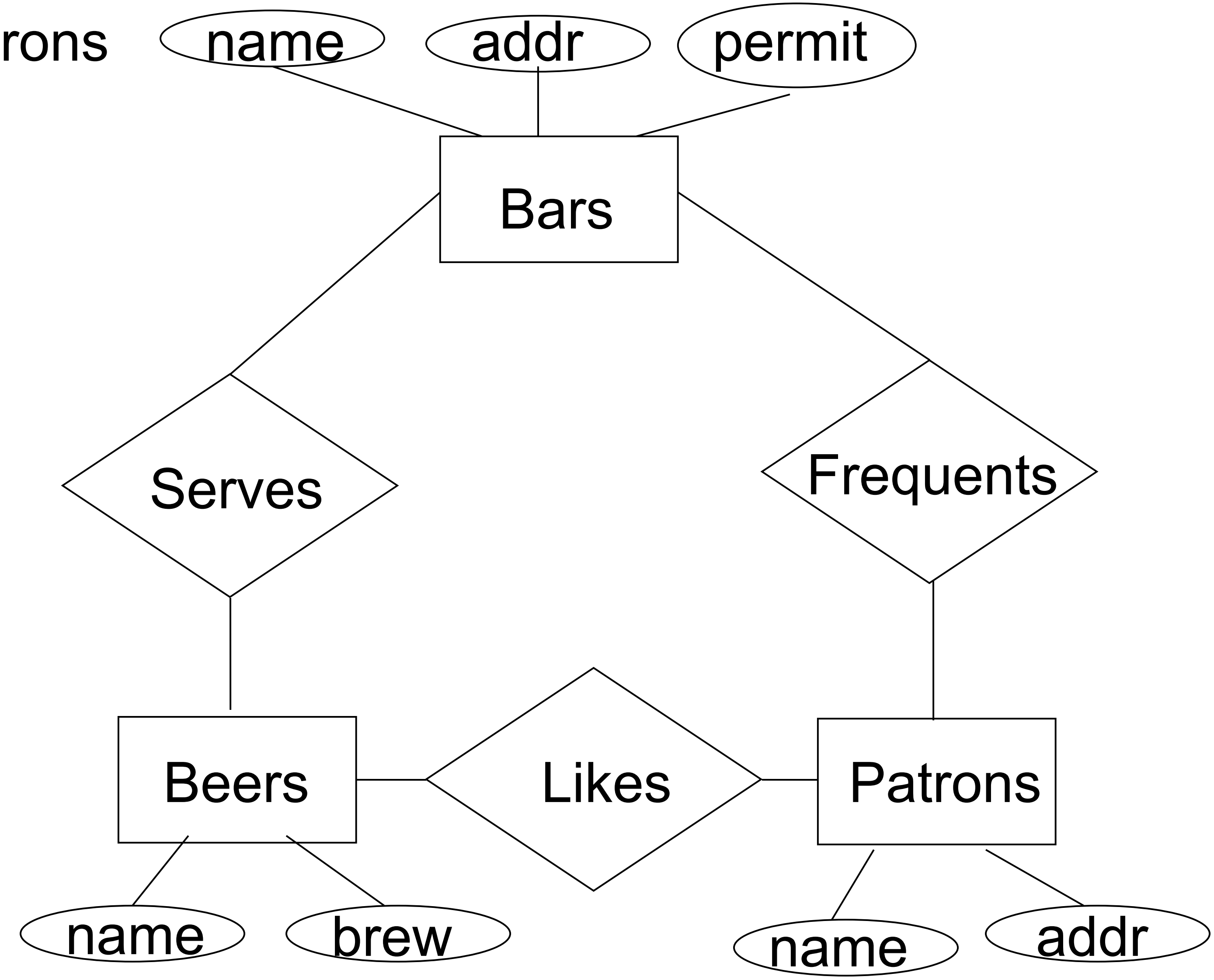
- A class may have **more than one** subclass.
- A class may have more than one class from which it inherits properties; those classes are its superclasses
- Subclasses may themselves have subclasses, yielding a **hierarchy** of classes where each class inherits the properties of its ancestors.

Multiple Inheritance in ODL



```
class MurderMystery extends Movie {
    attribute string weapon;
    };
    class CartoonMurderMystery extends Cartoon : MurderMystery;
```

Beers-Bars-Patrons



```
class Beers {
attribute string name;
attribute string manf;
relationship Set<Bars> servedAt
inverse Bars::serves;
relationship Set<Patrons> fans
inverse Patrons::likes;
}
```

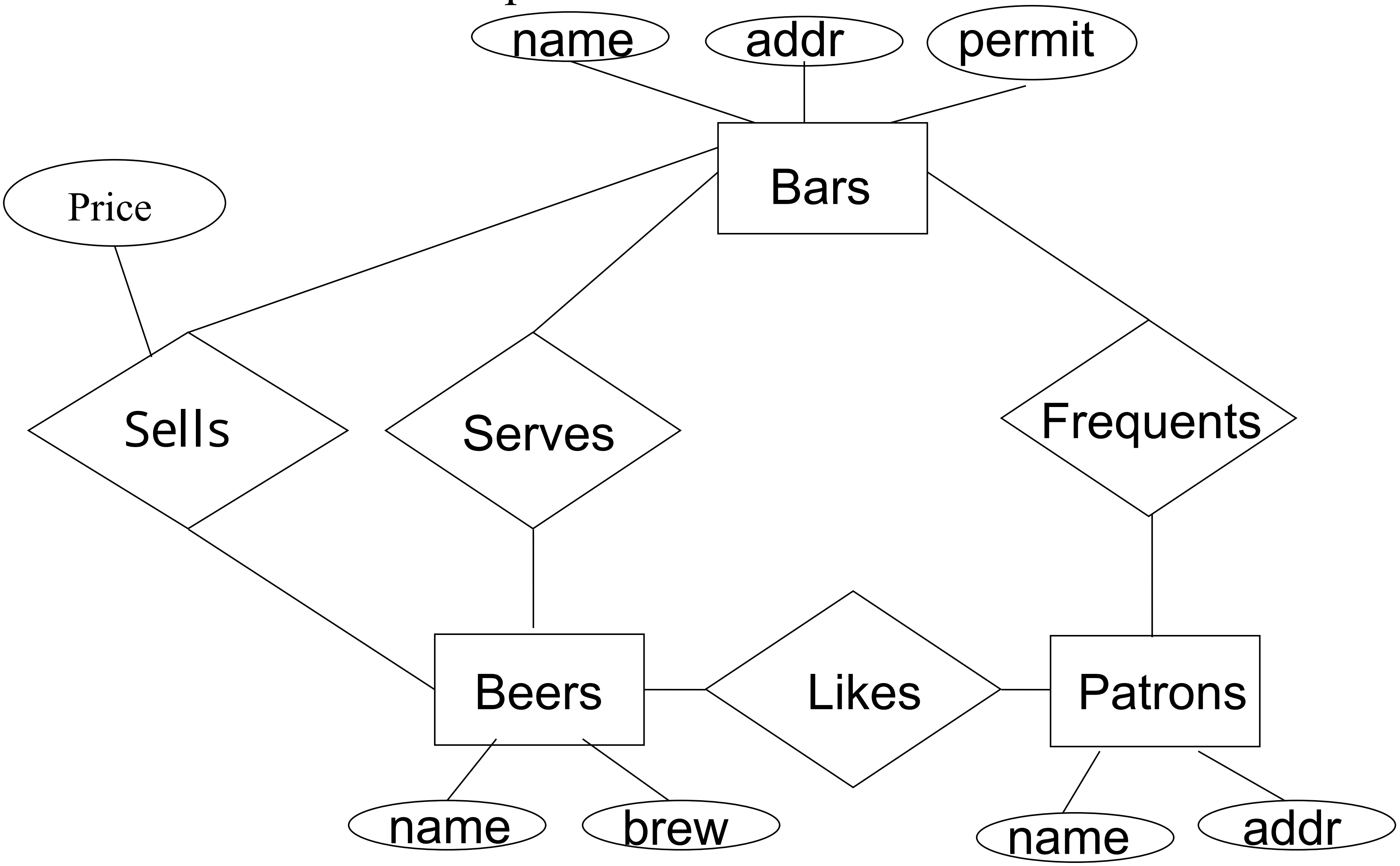
```
class Bars {
attribute string name;
attribute Struct Addr
{string street, string city, string PC}
address;
attribute Enum SAQ {full, beer,
BYOB,none} PermitType;
relationship Set<Patrons> customers
inverse Patrons::frequents;
relationship Set<Beers> serves
inverse Beers::servedAt;
}
```

Name is given to structure & enumeration type for possible reuse

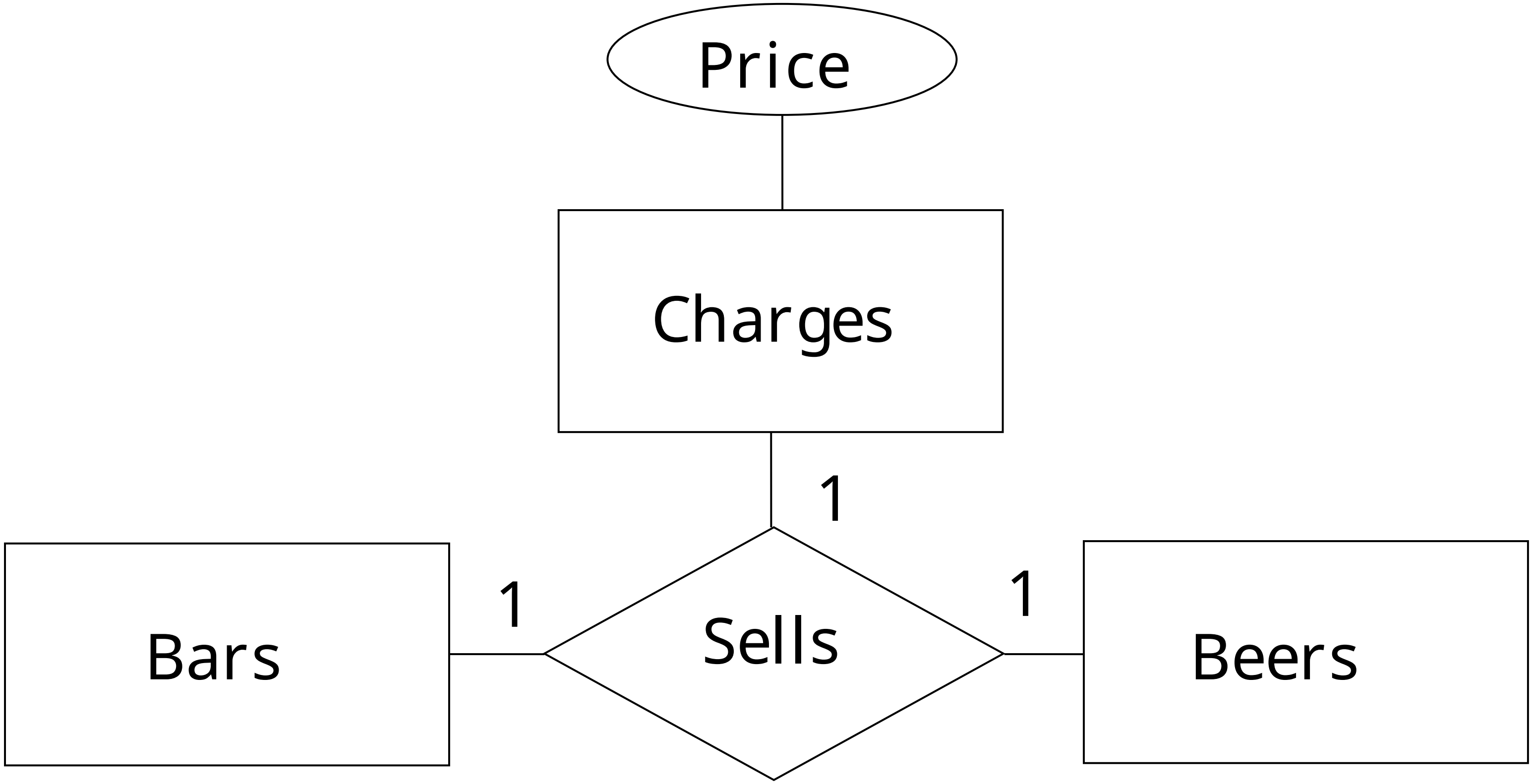
```
class Patrons {
attribute string name;
attribute Struct Bars::Addr
address;
relationship Set<Beers> likes
inverse Beers::fans;
relationship Set<Bars> frequents
inverse Bars::customers;
}
```

Reuse – qualify the name with the class for disambiguation

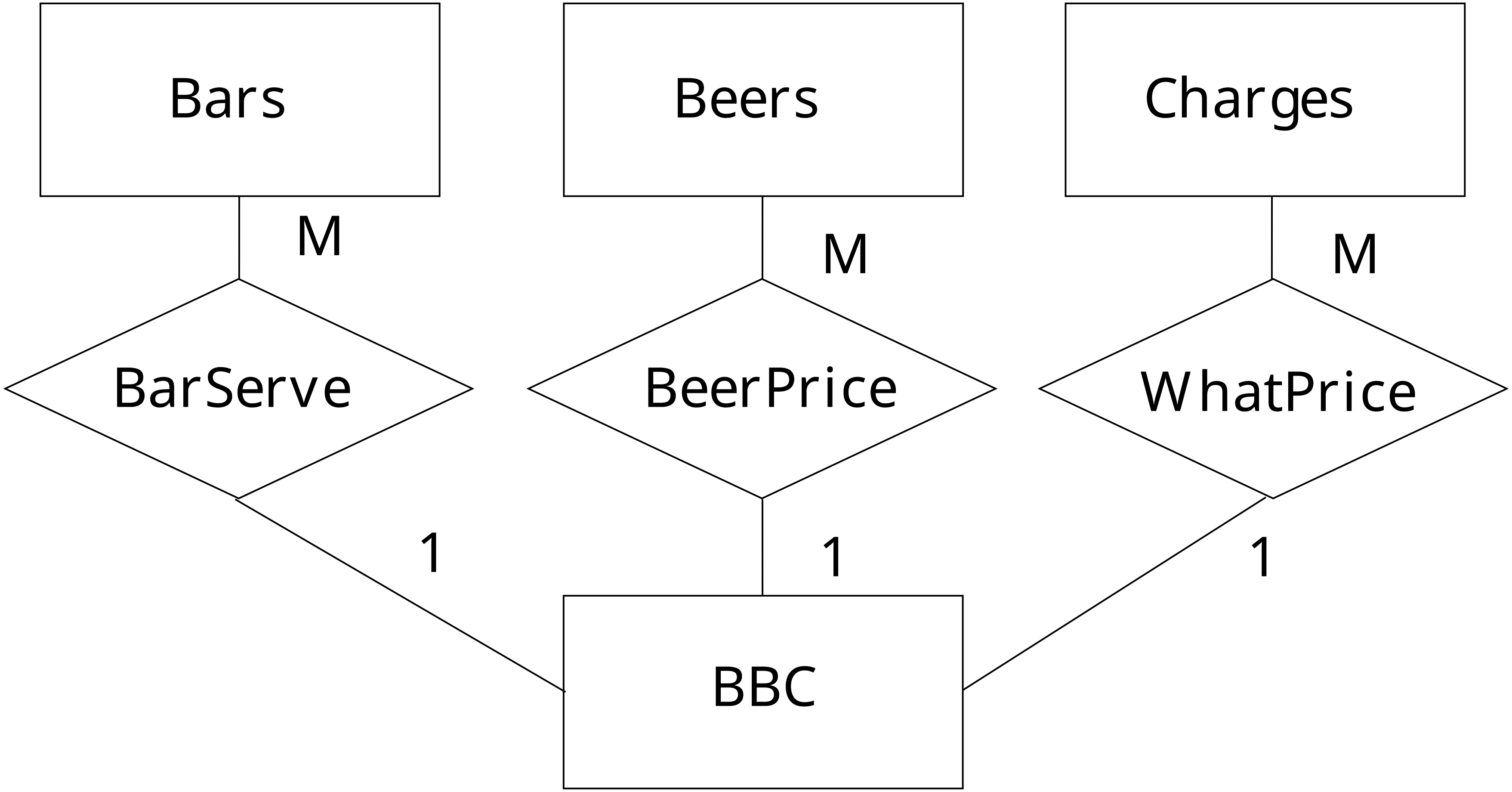
Attributes of Relationships



The attribute of a relationship converted into a three-way relationship!



If price depended only on the beer, then we could use two binary relationships: charge-beer and beer-bar.



(GobletOr, BrewXXX, $4.50)
(GobletOr, BrewX, $4.25)
(CanuckShack, BrewX, $4.10)

```
class Charges {
attribute real price;
relationship Set<BBC> HowMuch inverse BBP::WhatPrice;
}
class BBC {
relationship Bars BarServe inverse ...
relationship Beers BeerPrice inverse BeerCharges ...
relationship Charges WhatPrice  inverse Charges::HowMuch;
}
```

The same price may be charged at many bars!

Inverses must be added to Bars, Beers.

```
class Beers {
attribute string name;
attribute string manf;
relationship Set<Bars> servedAt   inverse Bars::serves;
relationship Set<Patrons> fans inverse Patrons::likes;
relationship Set <BBC> BeerCharges inverse BBC::BeerPrice }
```

From the internet to the Web

Early 80s: Archie, Veronica; internet file sharing finding systems

Late 80s early 90s

HTTP/HTML Tim Berners-Lee, Robert Caillau

Text based browser, Lynx, start of Netscape

HTTP request-response protocol between a client and a server

HTTP session is a sequence of requests-responses

HTML - very simple text markup language

included features for simple formatting and display

HTML based on ideas existing in the late 1980s including:

TeX/LaTex, Troff,

SGML and

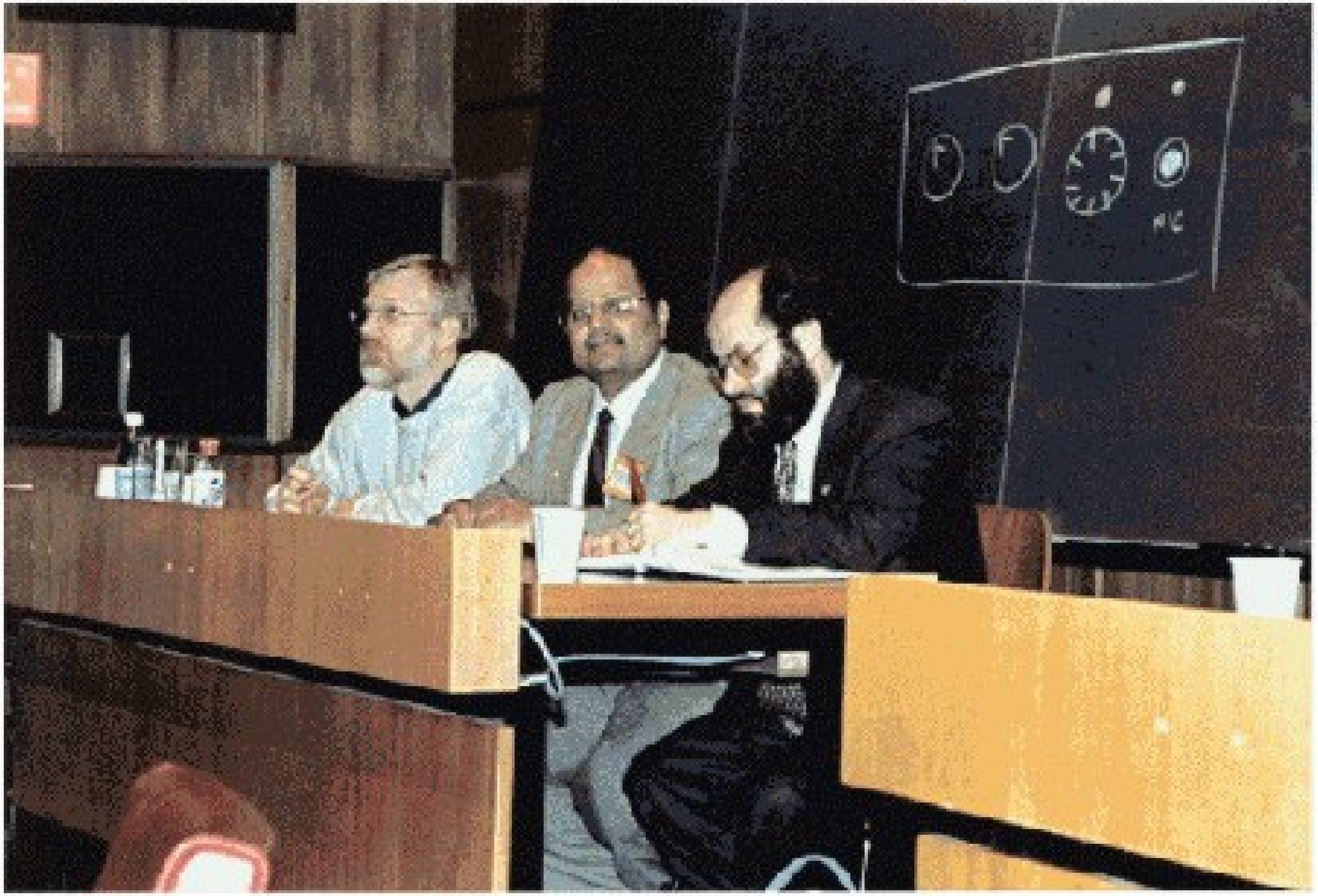
the early word processing software

(WordStar, WordPerfect, Word)

May 1994 CERN – Geneva: The first World Wide Web conference

Navigation Workshop for the Web

WWW94 - May 1994



SGML and HTML are mark up languages for information(textual)

HTML was too simple and not extensible

SGML was extensible but too complex.

Hence XML
E**X**tendible
Markup
Language

None of these languages do anything: none of these languages are ***Turing complete***

They provide a way to present information which is wrapped in tags. Note the evolution of data/metadata: metadata in program; metadata in schema; metadata with the data

The tags are commonly accepted by a community/group who want to exchange information.

SGML and XML specify the content and structure of a document in a way that allows particular presentations to be generated as needed

```
<!doctype linuxdoc system>
<!--   This a sample SGML file. Comments can appear anywhere It can
go over a number of lines. -->
<article>
<!-- Article type document -->
<title>Sample SGML Document
<!--  Always give a Title.  Should be descriptive -->
<author>Bipin C. DESAI
<date>March 2000
<!--  Note the tag minimization   the end tags are assumed by the
occurrence of a new tag  -->
<abstract>
This document is a sample document using the simple Linuxdoc-
SGML DTD: used to write all documents for Linux. There are other
DTDs and you can create your very own DTD. However, you have to
create all the scripts for its translation to other formats.
</abstract>
```

A sample DTD   saved as notes1.dtd

```
<!ELEMENT xslNotes   - O (title,author,para+)>
<!ELEMENT title        - O CDATA>
<!ELEMENT  author    - O CDATA>
<!ELEMENT para        - O CDATA>
```

O optional end tag
CDATA character data

Use of the sample DTD

```
<?xml version="1.0"?>
<!DOCTYPE xslNotes SYSTEM "notes1.dtd" >
 <xslNotes>
 <title>XSL Notes</title>
 <author>Bipin C. Desai</author>
 <para> This is  paragraph 1.
 <para> This is  paragraph 2.
 </xslNotes>
```

Why separate content and structure from presentation and behavior

Once coded, the information can be reused in many formats

Device/Media-independent publishing

One-on-one marketing

Intelligent downstream document processing

Large-scale information management.

XML (Extensible Markup Language): A subset of SGML (ISO 8879) designed for easy implementation

Information in XML form has to be rendered using appropriate formatting mechanism

XML document contains the syntax,

tags are used to provide “keys”

content within the tags represent the “value”

Tags have no predefined meaning but is agreed to by parties involved in the exchange of information

XML by itself conveys only content and structure, not presentation or behavior

XML data is stored in plain text format
independent of software/hardware.
makes it easy to share data

XML Simplifies Data Interchange

XML applications are designed and adapted to read xml data.

XML Simplifies Platform Changes

New platforms are designed/built so that they can use existing and new XML data

Since all new appliances implement XML features, XML data can be used with diverse devices

------- is fresher because more people eat it,
more people eat it because ------- is fresher!

XQuery is to XML what SQL is to relational databases.

XQuery was designed to query XML data.

XQuery for XML is what SQL for databases
XQuery is built on XPath expressions
XQuery is supported by all major databases

Path Expressions (no joins!)
XQuery uses path expressions to navigate through elements in an XML document.
XPath is used to address (select) parts of documents using **path expressions**
A path expression is a sequence of document step/tags separated by "/"

Each step operates on the set of instances produced by the previous step
Selection predicates may follow any step in a path, in []

XML Schema

The XML schema defines:

what are the components(elements) in a corresponding document

Order of these elements

Number of occurrences

Element's contents – could it be empty or it is required and its contents

Data types, default values

```
<?xml version="1.0"?>
<nns:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<nns:element name=“memo">
  <nns:complexType>
    <nns:sequence>
      <nns:element name="to" type="nns:string"/>
      <nns:element name="from" type="nns:string"/>
      <nns:element name=“subject" type="nns:string"/>
      <nns:element name="body" type="nns:string"/>
    </nns:sequence>
  </nns:complexType>
</nns:element>

</nns:schema>
```

Title	Authors array	Publication (Name, Vol, Date, pages)	Meeting	Subject set
Report of the Priorities Workshop	**[Caillau, Desai]**	(Computer Networks and ISDN Systems; Vol. 27-2,;November 1994; pp. 334-336)	WWW-I	{Web, searching}
Three Paradoxes of Big data	[Richards, King]	(Stanford, CA,;;Sept, 2013,;pp102-1050)	Making End Meet	{Big Data, security, privacy]

A not too correct XML schema to express this type of information is given in the next slide.

Exercise: complete the xml schema and create the xml doc for the above data!

```
<pns:schema xmlns:xs=“http://www.w3.org/2001/XMLSchema”>
<pns:element name=“papers” type=“Publications” />
<pns:element name=“tittle”>
    <pns:complexType>
        <pns:sequence>
            <pns:element name=“usual_title” type=“pns:string”/>
            <pns:element name=“alt_title” type=“pns:string”/>
       </pns:sequence>
    </pns:complexType>
</pns:element>
…………..
<pns:complexType name=“Publications”>
    <pns:sequence>
       <pns:element ref=“title” minOccurs=“0” maxOccurs=“unbounded”/>
       <pns:element ref=“authors” minOccurs=“0” maxOccurs=“unbounded”/>
       <pns:element ref=“publication” minOccurs=“0” maxOccurs=“1”/>
       <pns:element ref=“meeting” minOccurs=“0” maxOccurs=“1”/>
      <pns:element ref=“subjects” minOccurs=“0” maxOccurs=“unbounded”/>
    </pns:sequence>
</pns:complexType>
</pns:schema>
```

Predicates (where clause)
XQuery uses predicates to limit the extracted data from XML documents

Storing XML data
BLOB
Decompose and save as tables

Course Notes
Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

This document in electronic form, bearing a CopyForward permission, could be used for personal use and/or study, free of charge. Anyone could use it to derive updated versions. The derived version must be published under CopyForward. All authors of the version used to derive the new version must be included in the updated version in the existing order, followed by name(s) of author(s) producing the derived work

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

Storage Devices, Files, and Indexing

Bipin C. Desai

Storage Device Selection Criteria

Capacity vs. cost (What will $100 buy?
How much for 1 Megabytes?)

Cost per megabytes of storage has taken a plunge
Alas, the need for it has bounded as well.

Permanence

Portability

Relative cost

Performance (Latency, transfer /access rate)

Record size - buffer size, file size.

Accessing method - random/direct or sequential

Data transfer rate

Seek time - time to move read/write head:
average, minimum, maximum

Latency - rotational delay (rpm)

Memory Hierarchy

Speed	Technology	**Application**
1-10's nsec	I^2L	fast cache
	nmos	high speed MM
	bipolar	buffer
100's nsec	nmos	main memory
	core	
100's μsec	CCD	fast back up
	bubbles	
1-10's msec	floppy disk	main back up
	fixed head disk	
	moving head disk	
10's msec	magnetic tape	security/back up
100s of ms	optical memory	large mass
	tape library	memory,
	system	archives

Data on External Storage

Disks: Can retrieve random page at fixed cost

- But reading several consecutive pages is much cheaper than reading them in random order

Tapes: Can only read pages in sequence

- Cheaper than disks; used for archival storage – extinct??

File organization: Method of arranging a file of records on external storage.

- Record id (RID) is sufficient to physically locate record
- Indexes are data structures that allow us to find the record ids of records with given values in index search key fields

Architecture: Buffer manager stages pages from external storage to main memory buffer pool. File and index layers make calls to the buffer manager.

Store the database in Main Memory!

Costs too much. $100 will buy 32GB of DRAM or 1TB SSD today.

Main memory is volatile. We want data to be saved between runs. (Obviously!)

Typical storage hierarchy:

- Cache
- Main memory (RAM) for currently used data.
- Disk for the main database (secondary storage).
- Tapes for archiving older versions of the data (tertiary storage).

EXTERNAL STORAGE MEDIUMS

Read/Write

Magnetic Tape
Disk
RAID(Redundant array of inexpensive disks

Write once read many times(WORM) (used for archives)

Disks/tapes
Robotic storage media
CD-Rom.

READ: transfer data to main memory.
WRITE: transfer data to external device.
READ/WRITE are much slower than main-memory operations!

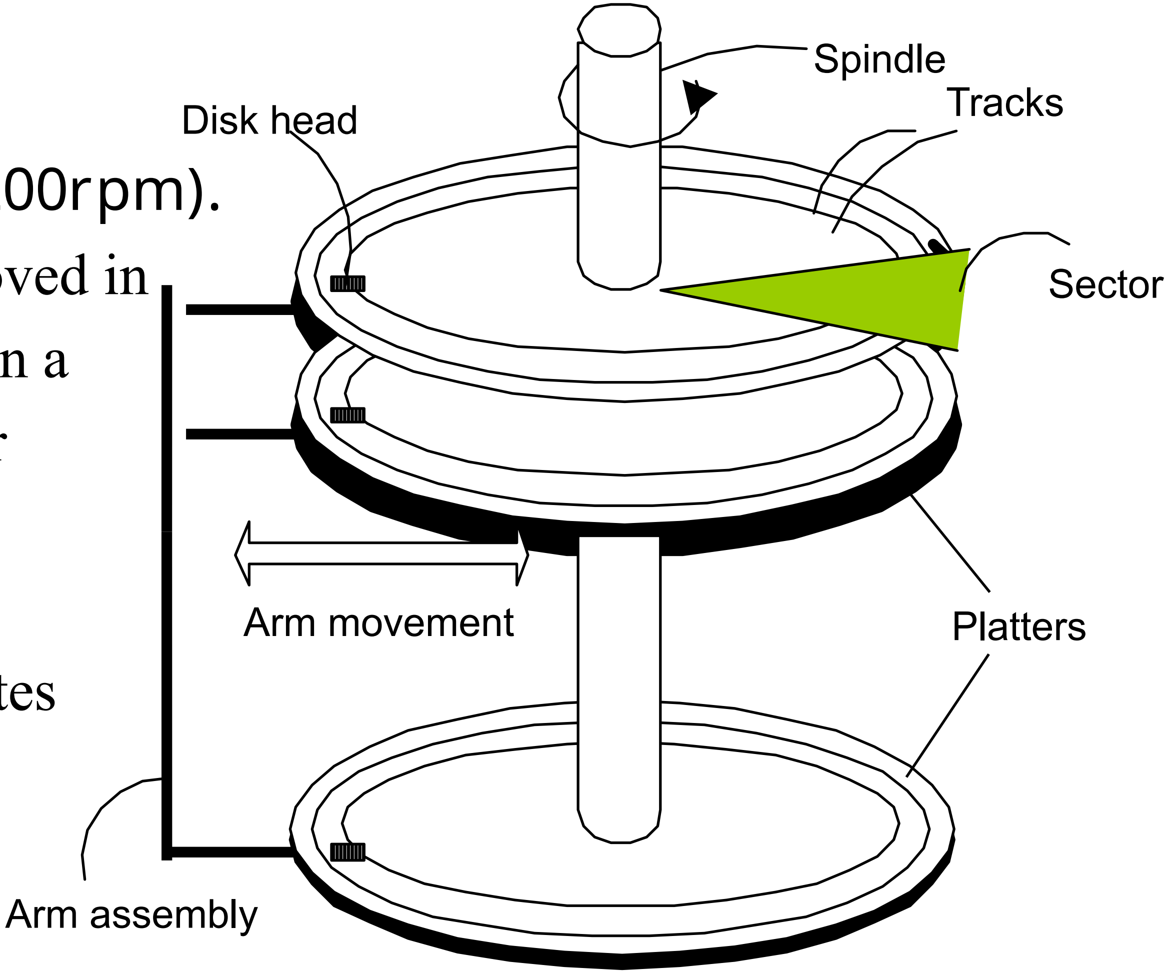
Disks

- Secondary storage device of choice:

  HDD are being replaced by SSD.
- Main advantage over tapes: *random access* vs. *sequential*.
- Data is stored and retrieved in units called *disk blocks* or *pages*.
- Unlike RAM, time to retrieve a disk page varies depending upon location on disk.
  - Therefore, relative placement of pages on disk has major impact on DBMS performance!

Components of a Hard Disk

- ❖ The platters spin (7200rpm).
- ❖ The arm assembly is moved in or out to position a head on a desired track. Tracks under heads make a *cylinder* (imaginary!).
- ❖ Only one head reads/writes at any one time.
- ❖ *Block size* is a multiple of fixed *sector size*



SSDs do not have moving parts but a finite number of cycles

Seek Time

Seek Time = $c_1 + c_2 *$ (number of cylinders to be traversed).

Here c_1 and c_2 are constants for a given model of disk drive.

Average Seek Time = time to move over 1/3 cylinders.

Seek time can be reduced by:

- distributing a file over a number of disk units and
- limiting the range of cylinders on any disk unit.

Rotational Latency

The delay between the completion of the seek and the actual transfer of data.

For a disk rotating at r (RPM)

$$t_1 = \frac{60 * 1000}{2* r} \text{ milliseconds}$$

RPM	Latency
10000	3 msec
7200	**4.1 msec**
6000	5 msec
3000	10 msec
2400	12.5 msec

Accessing a Disk Page

- Time to a read or a write a disk block:
  - *seek time* (moving arms to position disk head on track)
  - *rotational delay* (waiting for block to rotate under head)
  - *transfer time* (actually moving data to/from disk surface)
- Seek time and rotational delay dominate.
  - Seek time varies from about 1 to 20msec
  - Rotational delay varies from 0 to 10msec
  - Transfer rate is about 1msec per 4KB page
- Key to lower I/O cost: reduce seek/rotation delays! Hardware vs. software solutions?

SSD obsoletes these!

Response time = seek time + latency time + transfer time
(5-20 msec) (3-5 msec).

Transfer time = size of transfer/rate of transfer.

Size of transfer corresponds to the data of interest
(excluding format information, etc.)

Sequential Read of a number of blocks.

Transfer time = avg. seek time + latency time +
(block transfer time) * number of blocks
+ (min. seek time + latency) * number of cylinders

Problems: Disk scheduling in multi-process environment

Approximation: Transfer time = t_{efb} * # of blocks,

Here, t_{efb} is the effective formatted block transfer time.

$t_{efb} \approx 1.10 * t_b$, where t_b is the block transfer time
to account for the format information and the ignored seek and latency time.

Block transfer time = block size/ rate of transfer

Random Read of a # of blocks

Transfer time = number of blocks * (seek + latency + t_b)

Sequential Read from a number of contiguous cylinders

Transfer time = seek time +latency time + t_{efb} * # of block +
(min seek time + latency time) * (# of cylinders -1)

File Organisation

sequential
indexed sequential
direct access
other method

the storage required for the file organization
the time required to read a random record
the time required to read the next record
the time required to add a record
the time required to update a record
the time required to read all records
the time required to reorganize a file

Choice

- external storage device available
- use of the file - type of queries
- number of keys
- mode of retrieval - seq. random
- mode of update
- economy of storage
- frequency of use of a file
- growth potential of a file
- methods available in the development environment

simple
x = y
range
Boolean x=y
batch
on-line

Updates:

- insert in sequence
  at end, at first available location
- delete - compress first available location
  flag as deleted
- modify selected record space for update
  record size with respect to size of original record
- modify all records

Primary Key Retrieval

Four (three) possible choices -

- serial file - no order (pile)
- sequential - ordered wrt primary key
- indexed sequence
- direct access

Serial Files (PILE)

Access a random record

Access to Next Record

Inserting Record

Deleting a Record

Modifying a Record

Reorganisation

Single Disk Drive

Two or more Disk Drives

Sequential File

Access a random record

Access to Next Record

Inserting Record

Deleting a Record

Modifying a Record

Reorganisation

Single Disk Drive

Two or more Disk Drives

Access to Next Record

Probability of record in same block = $1 - 1/ b_f$

Probability of record not same block = $1/b_f$

Expected time to get next record.

$$= 0 * (1 - 1/ b_f) + 1 * (1/ b_f)*(t_s + t_l + t_b)$$

$$= 1/ b_f*(t_{efb})$$

Modify-in-place or Delete a Record

- Find it in T_f (Time to find random record)
- Max. time to modify or mark it as deleted, and wait $2T_l$ - block txf time
- Rewrite it in time = block txf time

Total time = $T_f + 2T_l$

Sector Addressable Disks

- fixed length arcs of a track - track is divided into an integral number of sectors.
- amount of data is fixed by O.S. or by the hardware.
  - simplifies allocation of storage space
  - simplifies address calculations
  - simplifies synchronisation of I/O & computation in sequential processing.

The division of a track into **sectors**:

-may be implemented completely by **hardware** or

- by **software** controlled formatting operation.

Block is a fixed number of bytes that is moved as a unit between storage devices and the main memory. Made up a number of disk sectors.

Arranging Pages on Disk

- `*Next*’ block concept:
  - blocks on same track, followed by
  - blocks on same cylinder, followed by
  - blocks on adjacent cylinder
- Blocks in a file should be arranged sequentially on disk (by `next’), to minimize seek and rotational delay.
- For a sequential scan, *pre-fetching* several pages at a time
- “De-fragmentation” to increase access

RAID

- Disk Array: Arrangement of several “inexpensive” disks that gives abstraction of a single, large disk.
- Goals: Increase performance and reliability.
- Two main techniques:
  - Data striping: Data is partitioned; size of a partition is called the striping unit. Partitions are distributed over several disks.
  - Redundancy: More disks => more failures. Redundant information allows reconstruction of data if a disk fails.

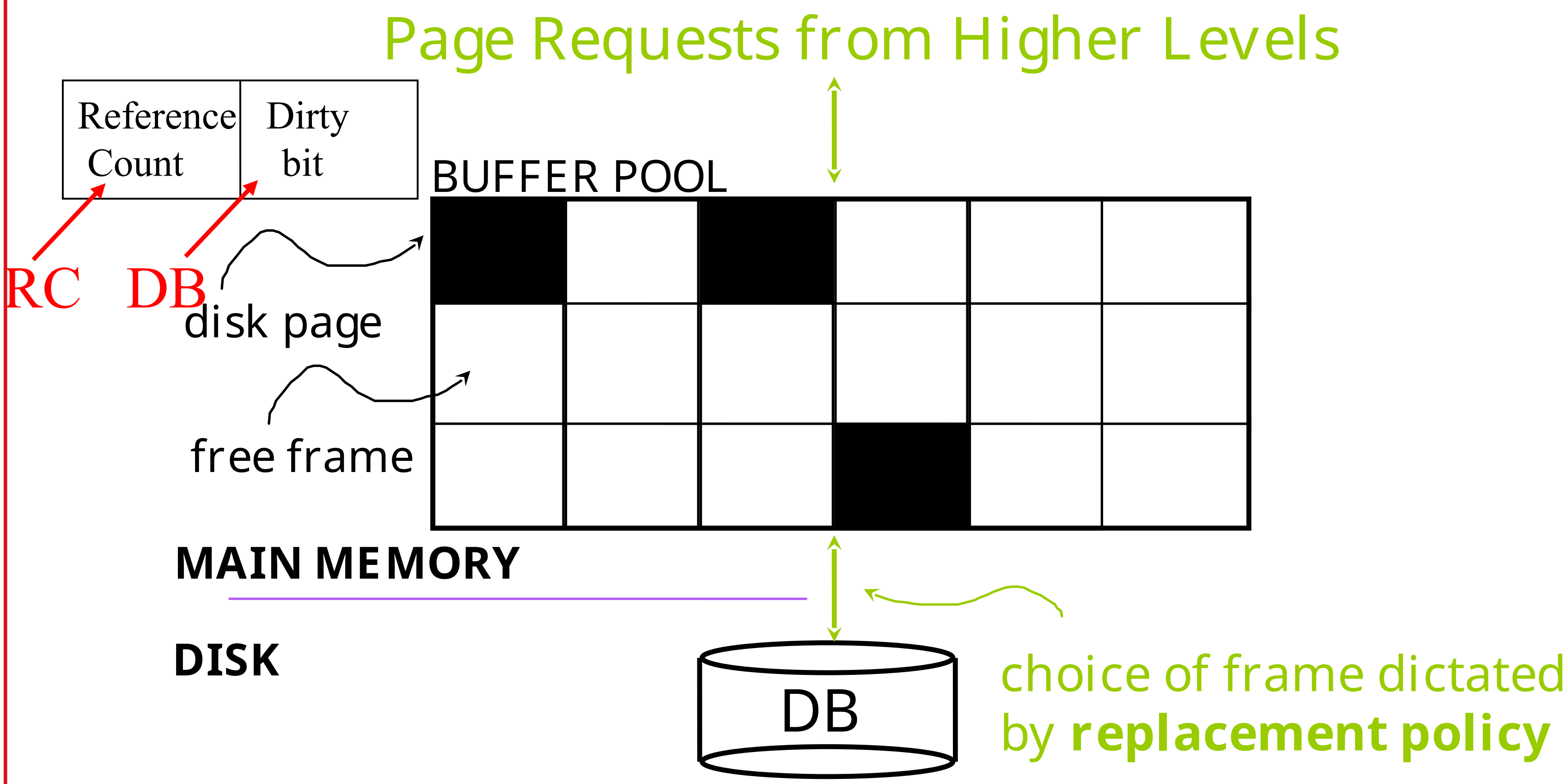
- Level 0: No redundancy
- Level 1: Mirrored (two identical copies)
  - Each disk has a mirror image (check disk)
  - Parallel reads, a write involves two disks.
  - Maximum transfer rate = transfer rate of one disk

- Level 0+1: Striping and Mirroring
  - Parallel reads, a write involves two disks.
  - Maximum transfer rate = aggregate bandwidth
- Level 3: Bit-Interleaved Parity
  - Striping Unit: One bit. One check disk.
  - Each read and write request involves all disks; disk array can process one request at a time.
- Level 4: Block-Interleaved Parity
  - Striping Unit: One disk block. One check disk.
  - Parallel reads possible for small requests, large requests can utilize full bandwidth
  - Writes involve modified block and check disk
- Level 5: Block-Interleaved Distributed Parity
  - Similar to RAID Level 4, but parity blocks are distributed over all disks

Disk Space Management

- Lowest layer of DBMS software manages space on disk.
- Higher levels call upon this layer to:
  - allocate/de-allocate a page
  - read/write a page
- Request for a *sequence* of pages must be satisfied by allocating the pages sequentially on disk! Higher levels don’t need to know how this is done, or how free space is managed.

DBMS: Buffer Management



Page Requests from Higher Levels

Reference Count | Dirty bit

RC DB

BUFFER POOL

disk page

free frame

MAIN MEMORY

DISK

DB

choice of frame dictated by **replacement policy**

- *DBMS operates on data in main memory*
- *Buffer management maintains a table <frame#, pageid>*

When a Page is Requested ...

- If requested page is not in pool:
  - Choose a frame for *replacement (LIFO, FIFO, LRU(RC), modified (DB), etc.)*
  - If frame is dirty (changed since read into buffer), write it to disk(replacement frame scheme looks for non-dirty frame
  - Read requested page into chosen frame
- *Increment the* ***reference count*** *(RC) of* the page and return its address.

If requests can be predicted (e.g., sequential scans) pages can be pre-fetched

More on Buffer Management

- When a frame is released by an application, the RC is decremented and if the frame is changed, the dirty bit for the frame is set.
- A frame in the buffer may be requested many times, concurrently(reads – not update/write)
  - a *RC* is used to indicate the number of concurrent use of a frame. A frame is a candidate for replacement iff $RC = 0$.
  - Priority if dirty bit is not set(not modified)
- Concurrency control and recovery may entail additional I/O when a frame is chosen for replacement.

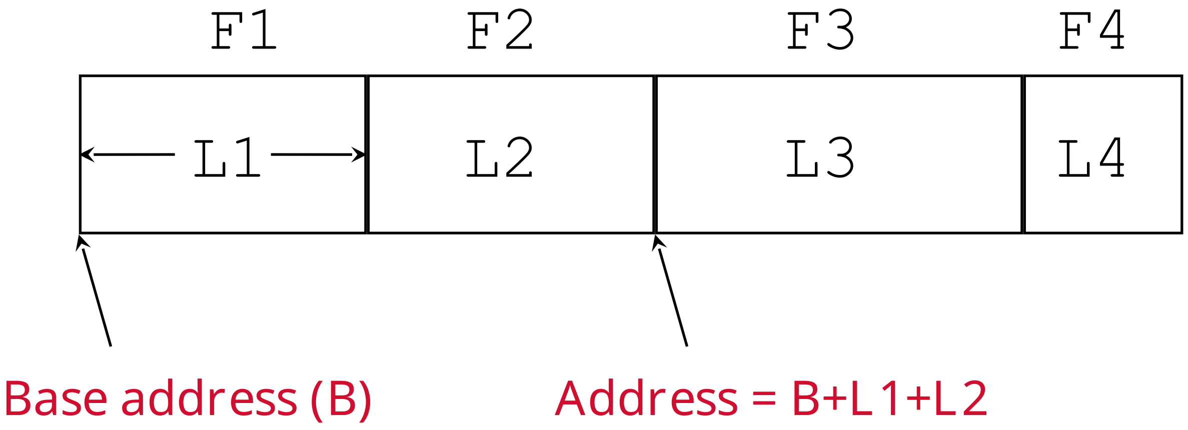
Buffer Replacement Policy

- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), Clock, MRU etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern.*
- *Sequential flooding*: Nasty situation caused by LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request could cause an I/O.

DBMS vs. OS File System

- Differences in different level of support in different OS: portability issues
- Some limitations, e.g., files can’t span disks.
- Buffer management in DBMS requires ability to:
  - Manage RC and DB of frames in buffer pool, force a page to disk (important for implementing concurrency control and recovery),
  - adjust *replacement policy,* and pre-fetch pages based on access patterns in typical DB operations.

Data Record Formats: Fixed Length



F1 F2 F3 F4

L1 L2 L3 L4

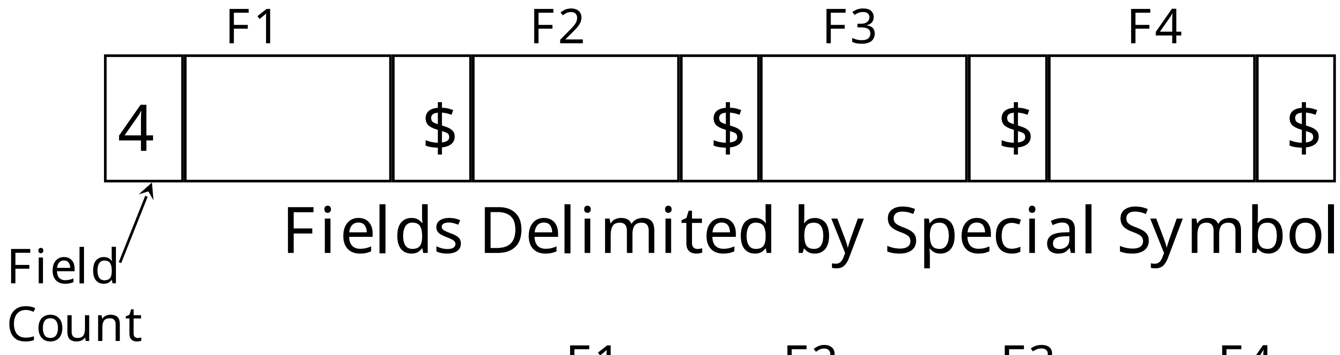
Base address (B)

Address = B+L1+L2

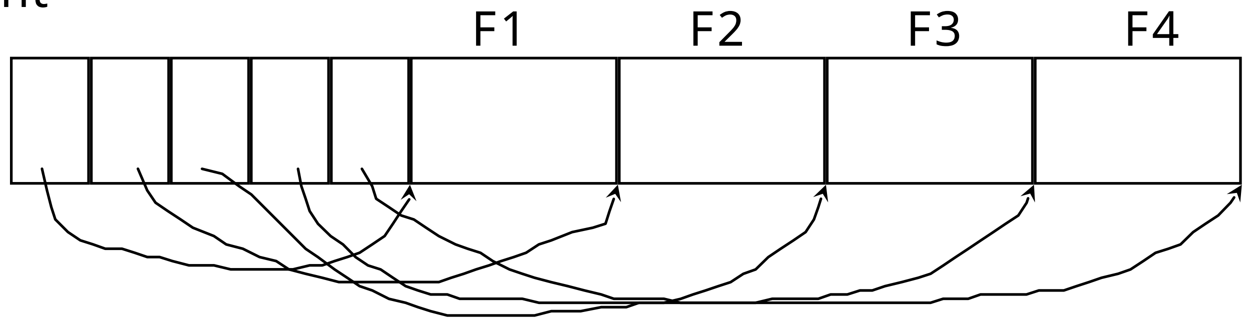
- Information about field types same for all records in a file; stored in *system catalogs.*
- Finding *i'th* field does not require scan of record.

Data Record Formats: Variable Length

- Two alternative formats (# fields is fixed):



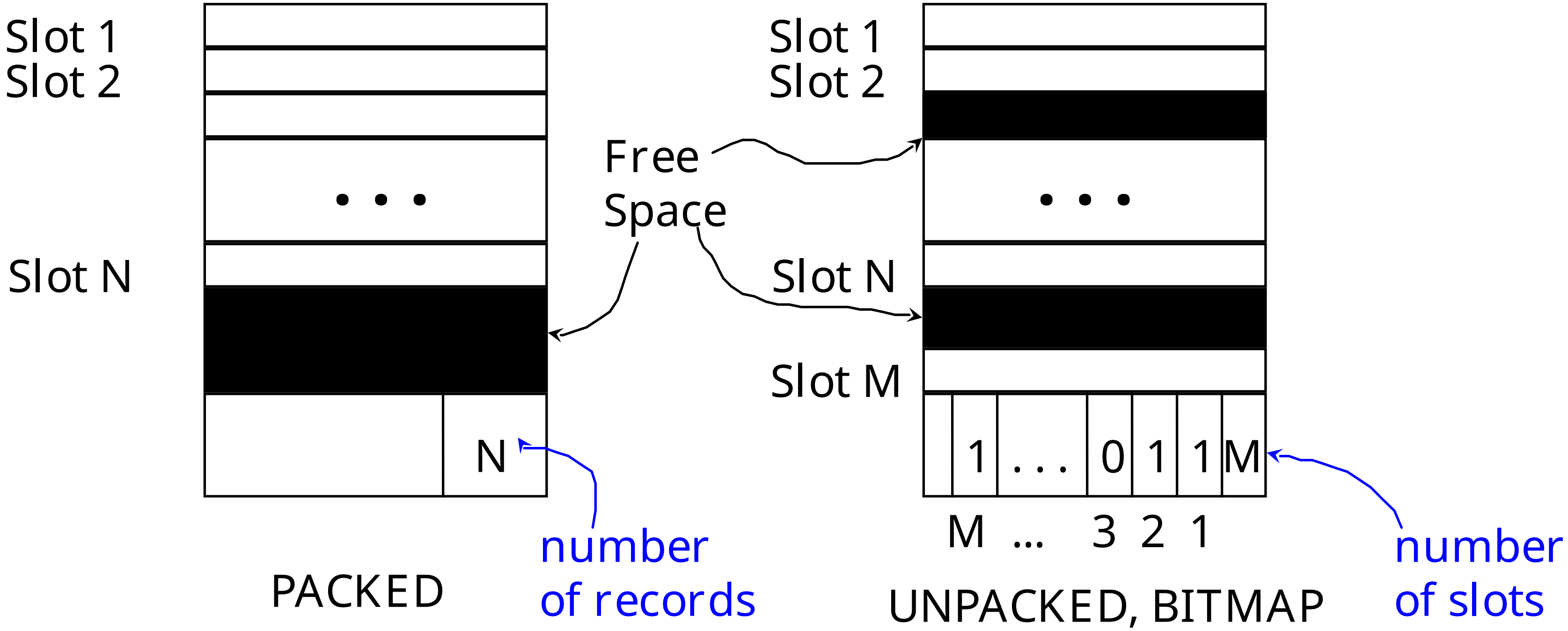
Fields Delimited by Special Symbols



Array of Field Offsets

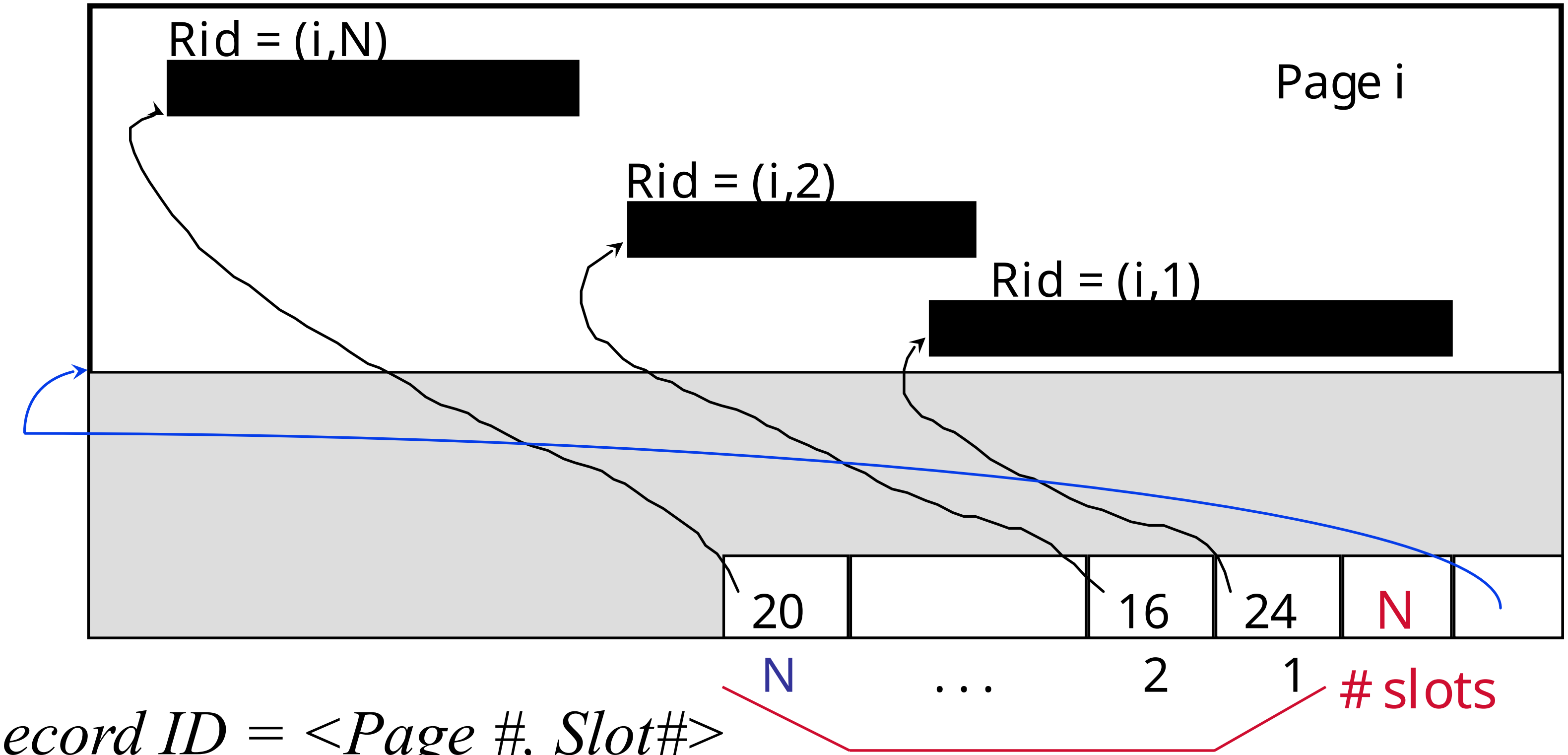
☛ Second offers direct access to i’th field, efficient storage of *nulls* (special *don’t know* value); small directory overhead.

Page Formats: Fixed Length Records



Record id = <page id, slot #>. In first alternative, moving records for free space management changes rid; may not be acceptable.

Page Formats: Variable Length Records



Record ID = <Page #, Slot#>

Slots contains address or offset of record

Can move records on the page without changing the record ID (RID);

Can also be used for fixed-length records!

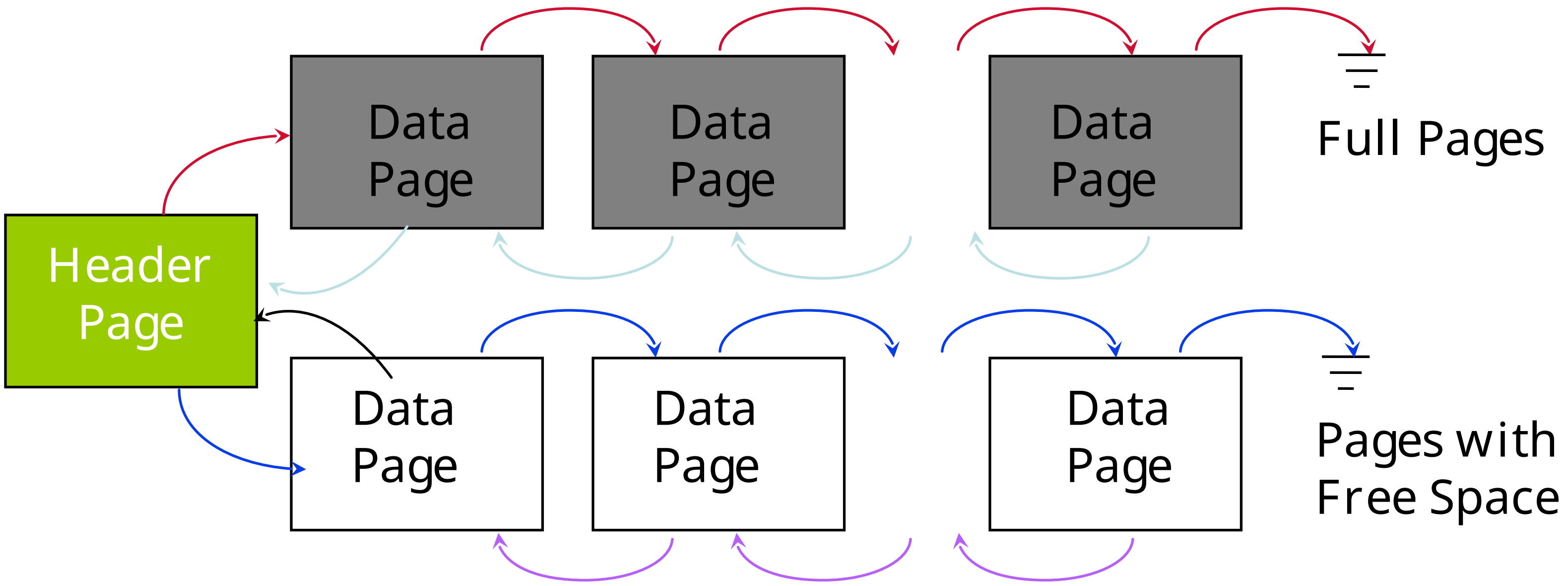
Files of Records

- Page or block is OK when doing I/O, but higher levels of DBMS operate on *records*, and *files of records*.
- FILE: A collection of pages, each containing a collection of records. Must support:
  - insert/delete/modify record
  - read a particular record (specified using *record id*)
  - scan all records (possibly with some conditions on the records to be retrieved)

Unordered (Heap) Files

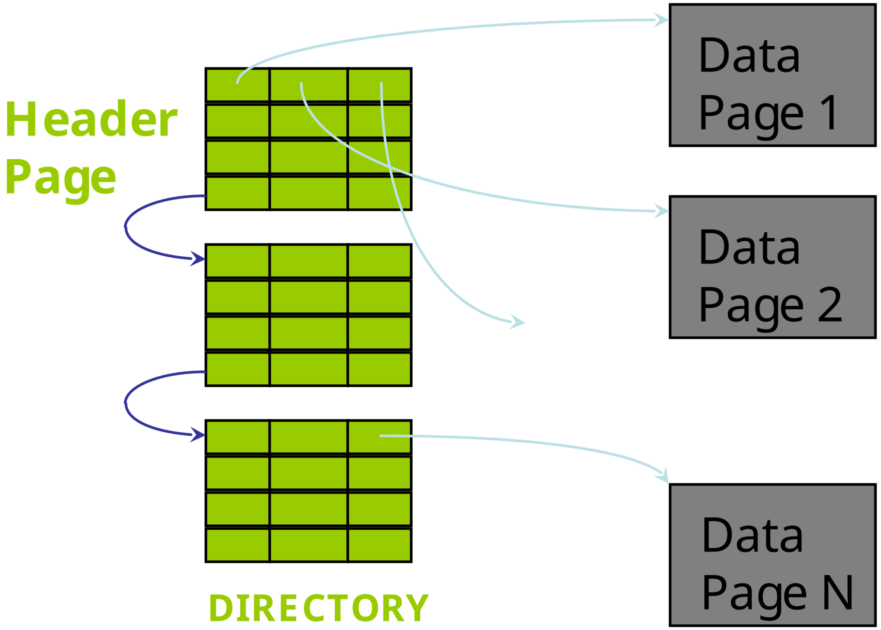
- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on a page
- There are many alternatives for keeping track of these details.

Heap File Implemented as a List



- The Heap file name and its header page address must be stored in a catalog.
- Each page contains 2 `pointers’ (forward, reverse) plus data.

Heap File Using a Page Directory



- The entry for a page can include the amount of free space on the page.
- The directory is a collection of pages; for example implemented as a linked list
- *Much smaller than linked list of all HF pages!*

System Catalogs

- *Catalogs are stored as tables.*
- For each table:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each view:
  - view name and definition
- For each index:
  - structure (e.g., B+ tree) and search key fields
- Plus statistics, authorization, buffer pool size, etc.

Alternative File Organizations

Heap files: Suitable when typical access requires access to all records in a file.

Sorted Files: Suitable in cases where the records must be retrieved in some order wrt a “key”, or access to a records in a `range’ of key values is needed.

Hashed Files: Suitable when random access to records with a given key value is required.

B: The number of blocks (pages) for data

b_f: Blocking factor(# records per block)

t_{efb} **:** Effective time to read or write block

	Heap File	Sorted File	Hashed File
Scan all recs	**B**t_{efb}	**B**t_{efb}	**1.25 B**t_{efb}
Equality Search	**0.5 B**t_{efb}	t_{efb} **$\log_2$B**	t_{efb}
Range Search	**B**t_{efb}	t_{efb} **($\log_2$B + # of blocks with matches)**	**1.25 B**t_{efb}
Insert	**2**t_{efb}	**Search + B**t_{efb}	**2**t_{efb}
Delete	**Search +** t_{efb}	**Search + B**t_{efb}	**2**t_{efb}

Hash 1.25: since pages are only 80% full for avoiding overflows

INDEX

An index is created to speed up access to the records in a file with a given value for a **search key fields**.

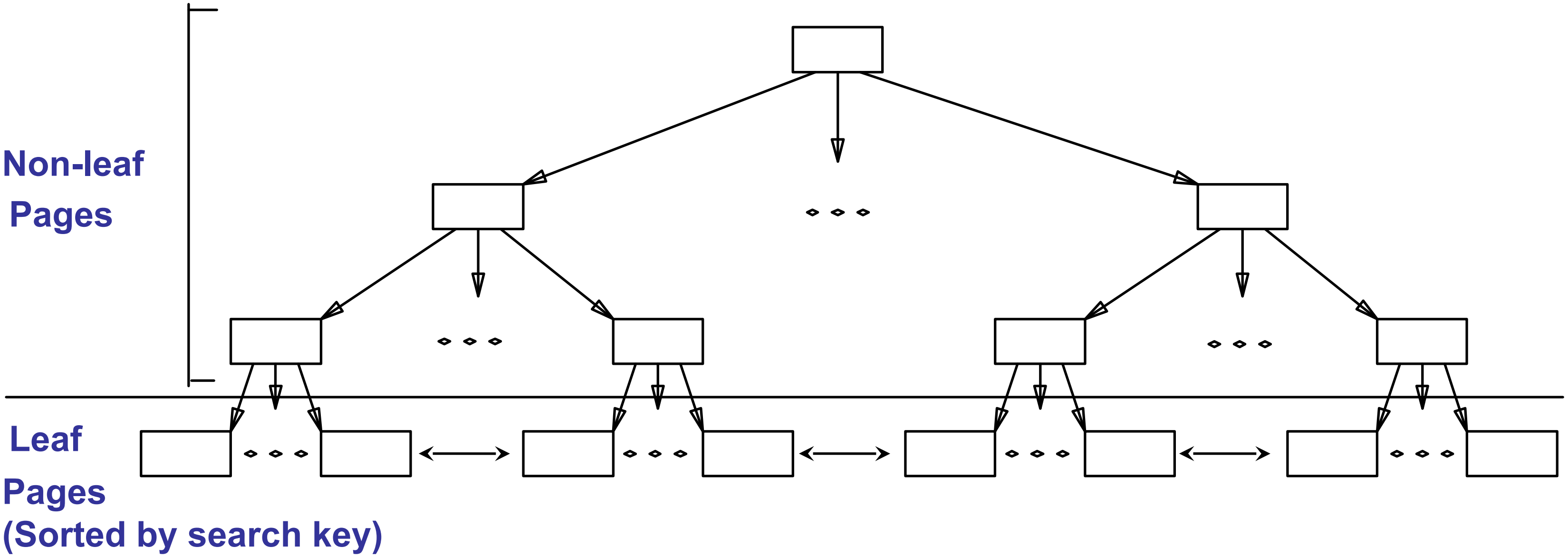
Any subset of the fields of a record can be used as search key for an index on the relation.

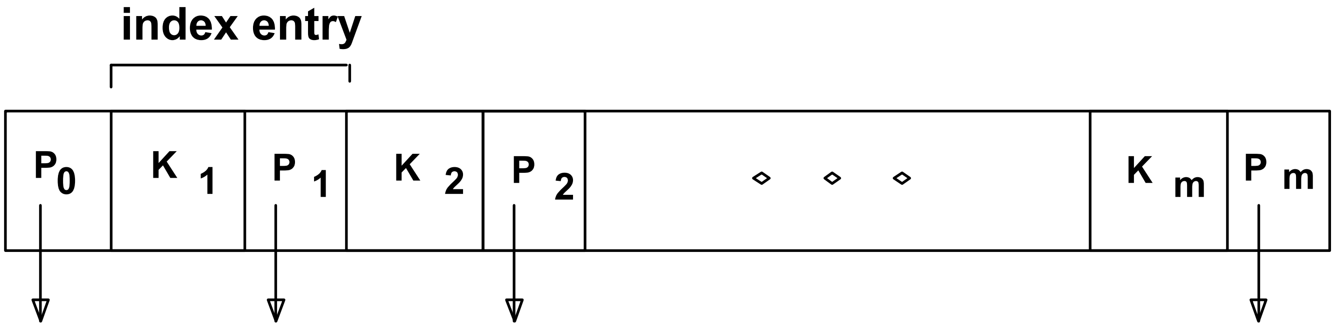
Search key may not be the same as primary key

An index contains a collection of data entries, and supports efficient retrieval of all records with a given search key value **K**.

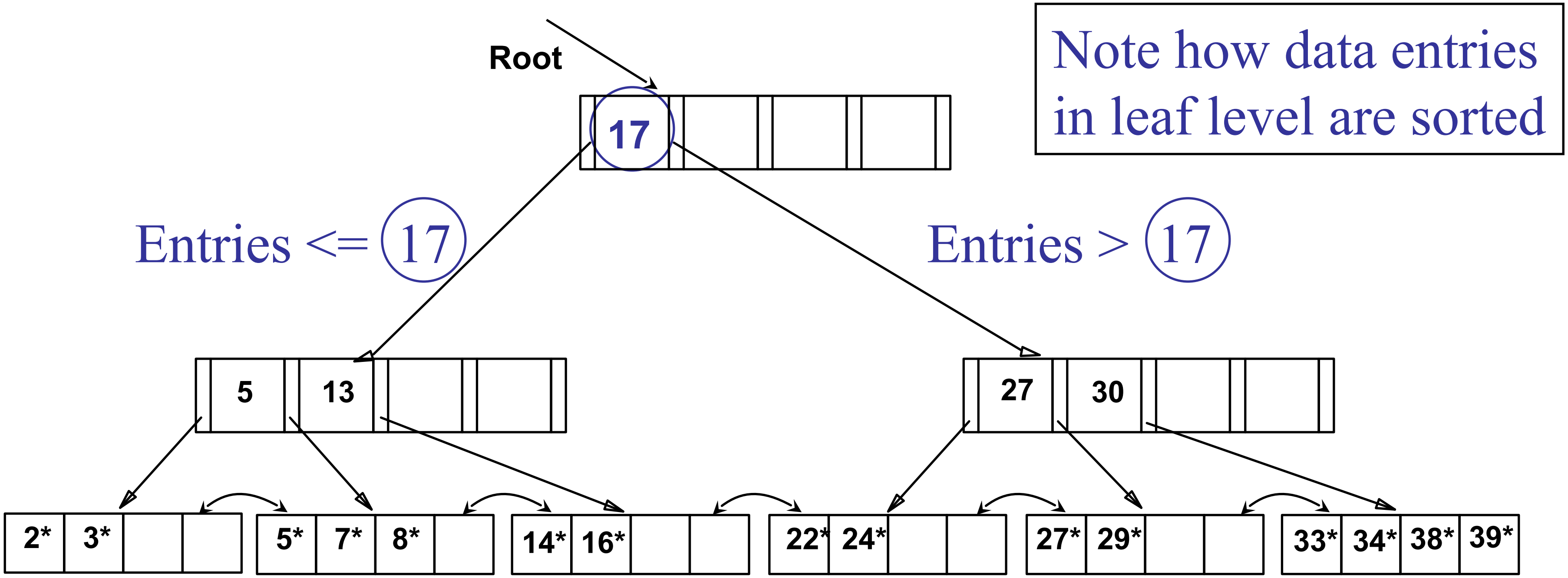
B+ Tree Indexes

- Internal nodes (pages) have *index entries;* only used for navigation:
- Leaf pages contain *data entries*, and are chained (prev & next)





Example B+ Tree



- Find 28*? 29*? All > 15* and < 30*
- Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

Hash-Based Indexes

- Good for equality selections.
- Index is a collection of *buckets*.
  - Bucket = *primary* page plus zero or more *overflow* pages.
  - Buckets contain data entries.
- *Hashing function* **h**: **h**(*r*) = bucket in which (data entry for) record *r* belongs. **h** looks at the *search key* fields of *r.*
  - *No need for "index entries" in this scheme.*

Alternatives for contents of an Index

- In an index entry k* we can store:

  Alternative 1: The actual data record with key value **k,** or

  Alternative 2: <**k**, rid of data record with search key value **k**>, or

  Alternative 3 <**k**, list of rids of data records with search key **k**>

- Choice of alternative for data entries is orthogonal to the indexing technique used to locate data entries with a given key value **k**.
  - Examples of indexing techniques: B+ trees, hash-based structures
  - Typically, index contains auxiliary information that directs searches to the desired data entries

Alternatives for Data Entries

- Alternative 1:
  - If this is used, index structure is a file organization for data records (instead of a Heap file or sorted file).
  - At most one index on a given collection of data records can use Alternative 1. (Otherwise, data records are duplicated, leading to redundant storage and potential inconsistency.)
  - If data records are very large, # of pages containing data entries is high. Implies size of auxiliary information in the index is also large.

Alternatives for Data Entries

- Alternatives 2 and 3:
  - Data entries typically much smaller than data records. Better than Alternative 1 with large data records, especially if search keys are small. (Portion of index structure used to direct search, which depends on size of data entries, is much smaller than Alternative 1.)
  - Alternative 3 more compact than Alternative 2, but leads to variable sized data entries even if search keys are of fixed length.

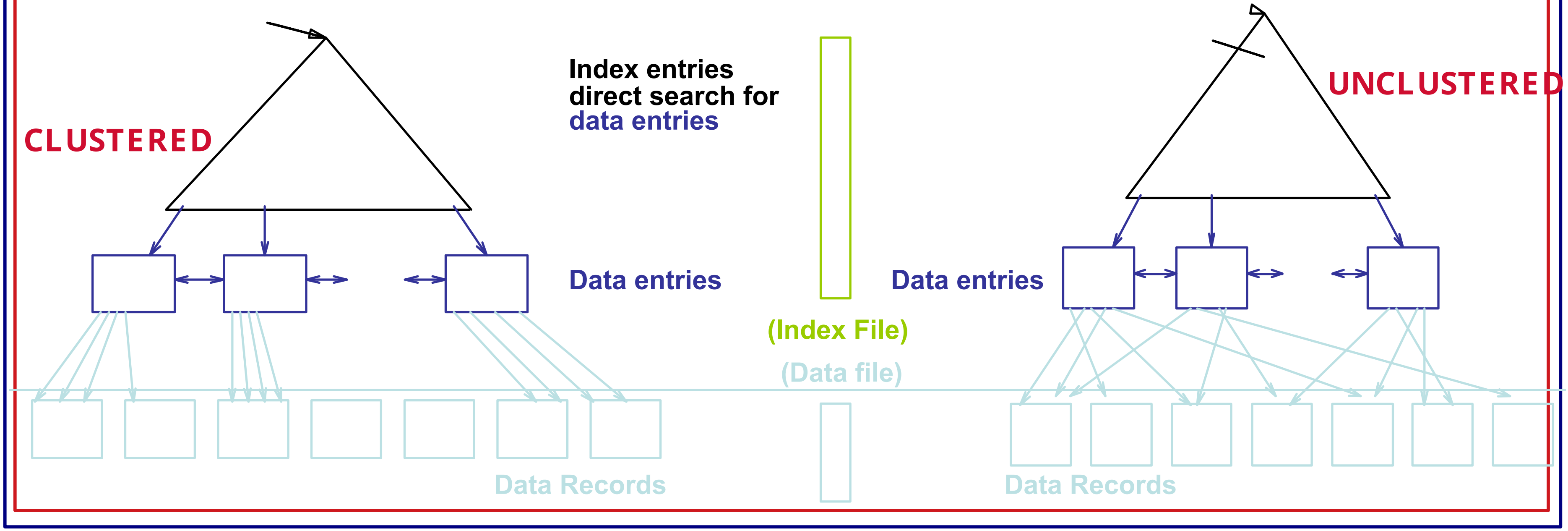
Index Classification

- *Primary* vs. *secondary*: If search key contains primary key, then it is called primary index.
  - *Unique* index: Search key contains a candidate key.
- *Clustered* vs. *un-clustered*: If the order of the data records is the same as, or `close to', the order of the data entries, then the index is called a clustered index: else un-clustered.
  - Alternative 1 implies clustered; in practice, clustered also implies Alternative 1 (since sorted files are rare).
  - A file can be clustered on at most one search key.
  - Cost of retrieving data records through index varies *greatly* based on whether index is clustered or not!

Clustered vs. Unclustered Index

Suppose that Alternative (2) is used for data entries, and that the data records are stored in a Heap file.

- To build clustered index, first sort the Heap file (with some free space on each page for future inserts).
- Overflow pages may be needed for inserts. (Thus, order of data recs is `close to’, but not identical to, the sort order.)



Cost Model for Our Analysis

We ignore CPU costs, for simplicity:

- **B:** The number of data pages
- **R:** Number of records per page
- **D:** (Average) time to read or write disk page
- Measuring number of page I/O's ignores gains of pre-fetching a sequence of pages; thus, even I/O cost is only approximated.
- Average-case analysis; based on several simplistic assumptions.

These are only estimates to show the overall trends!

Comparing File Organizations

- Heap files (random order; insert at eof)
- Sorted files, sorted on <*age, sal*>
- Clustered B+ tree file, Alternative (1), search key <*age, sal*>
- Heap file with un-clustered B + tree index on search key <*age, sal*>
- Heap file with unclustered hash index on search key <*age, sal*>

Operations to Compare

- Scan: Fetch all records from disk
- Equality search
- Range selection
- Insert a record
- Delete a record

Assumptions

- Heap Files: Equality selection on key; exactly one match.
- Sorted Files: Files compacted after deletions.
- Indexes: Alt (2), (3): data entry size = 10% size of record
  - Hash: No overflow buckets.
    - 80% page occupancy => File size = 1.25 data size
  - Tree: 67% occupancy (this is typical).
    - Implies file size = 1.5 data size
- Scans: Leaf levels of a tree-index are chained.
  - Index data-entries plus actual file scanned for unclustered indexes.
- Range searches:We use tree indexes to restrict the set of data records fetched, but ignore hash indexes.

Cost of Operations

B: The number of data pages
R: Number of records per page
D: (Average) time to read or write disk page

	(a) Scan	(b) Equality	(c) Range	(d) Insert	(e) Delete
(1) Heap	BD	0.5BD	BD	2D	Search +D
(2) Sorted	BD	$D \log_2 B$	$D(\log_2 B$ + # pgs with match recs)	Search + BD	Search +BD
(3) Clustered	1.5BD	$D \log_F 1.5B$	$D(\log_F 1.5B$ + # pgs w. match recs)	Search + D	Search +D
(4) Unclust. Tree index	BD(R+0.15)	$D(1 + \log_F 0.15B)$	$D(\log_F 0.15B$ + # pgs w. match recs)	Search + 2D	Search + 2D
(5) Unclust. Hash index	BD(R+0.125)	2D	BD	Search + 2D	Search + 2D

These are estimates using many simplifying assumptions

Understanding the Workload

- For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
- For each update in the workload:
  - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

Choice of Indexes

- What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- For each index, what kind of an index should it be?
  - Clustered? Hash/tree?
- One approach: Consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index. If so, create it.
  - Obviously, this implies that we must understand how a DBMS evaluates queries and creates query evaluation plans!
  - For now, we discuss simple 1-table queries.
- Before creating an index, must also consider the impact on updates in the workload!

Index Selection Guidelines

- Attributes in WHERE clause are candidates for index keys.
  - Exact match condition suggests hash index.
  - Range query suggests tree index.
    - Clustering is especially useful for range queries; can also help on equality queries if there are many duplicates.
- Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
  - Order of attributes is important for range queries.
  - Such indexes can sometimes enable index-only strategies for important queries.
    - For index-only strategies, clustering is not important!
- Try to choose indexes that benefit as many queries as possible. Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

Examples of Clustered Indexes

```
SELECT  E.dno
FROM  Emp E
WHERE  E.age>40
```

B+ tree index on E.age can be used to get qualifying tuples.

- How selective is the condition?
- Is the index clustered?

Consider the GROUP BY query.

- If many tuples have *E.age* > 10, using *E.age* index and sorting the retrieved tuples may be costly.
- Clustered *E.dno* index may be better!

```
SELECT  E.dno, COUNT (*)
FROM  Emp E
WHERE  E.age>10
GROUP BY E.dno
```

Equality queries and duplicates:

- Clustering on *E.hobby* helps!

```
SELECT  E.dno
FROM  Emp E
WHERE  E.hobby=Stamps
```

Indexes with Composite Search Keys

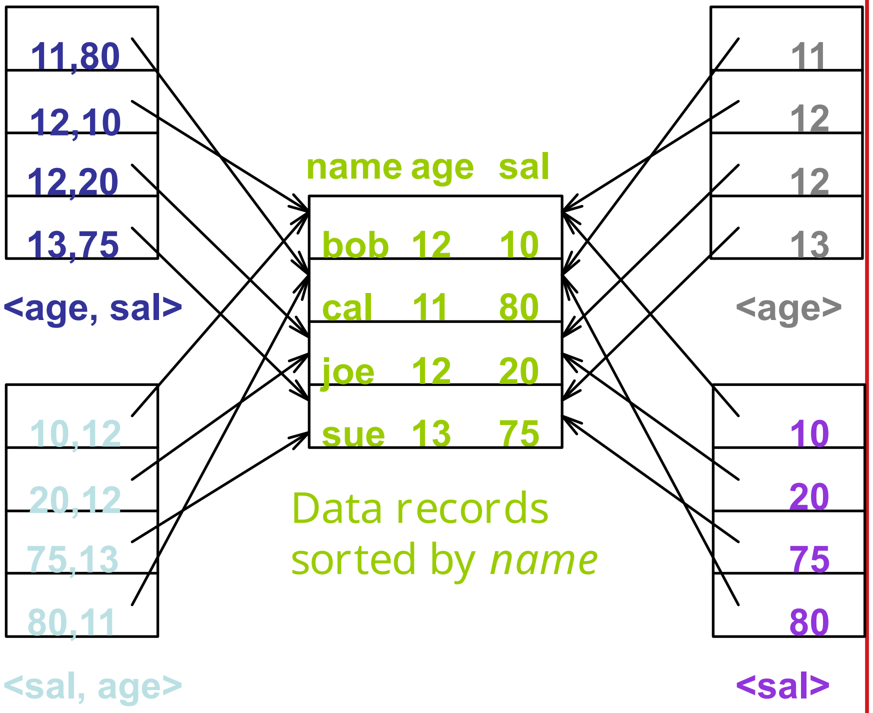
Composite Search Keys: Search on a combination of fields.

- Equality query: Every field value is equal to a constant value. E.g. wrt <sal,age> index:
  - age=20 and sal =75
- Range query: Some field value is not a constant. e.g.:age =20; or age=20 and sal > 10

Data entries in index sorted by search key to support range queries.

- Lexicographic order, or
- Spatial order.

Examples of composite key indexes using lexicographic order.



Data entries in index sorted by <*sal,age*>

Data entries sorted by <*sal*>

Composite Search Keys

- To retrieve Emp records with *age*=30 AND *sal*=4000, an index on <*age,sal*> would be better than an index on *age* or an index on *sal*.
  - Choice of index key orthogonal to clustering etc.
- If condition is: 20<*age*<30 AND 3000<*sal*<5000:
  - Clustered tree index on <*age,sal*> or <*sal,age*> is best.
- If condition is: *age*=30 AND 3000<*sal*<5000:
  - Clustered <*age,sal*> index much better than <*sal,age*> index!
- Composite indexes are larger, updated more often.

Index-Only Plans

- A number of queries can be answered without retrieving any tuples from one or more of the tables involved if a suitable index is available.

<E.dno>

```
SELECT  E.dno, COUNT(*)
FROM  Emp E
GROUP BY  E.dno
```

<E.dno,E.sal>

Tree index!

```
SELECT  E.dno, MIN(E.sal)
FROM  Emp E
GROUP BY  E.dno
```

<E. age,E.sal>

or

<E.sal, E.age>

Tree index!

```
SELECT AVG(E.sal)
FROM  Emp E
WHERE  E.age=25 AND
  E.sal BETWEEN 3000 AND 5000
```

Index-Only Plans (Contd.)

- Index-only plans are possible if the key is <dno,age> or we have a tree index with key <age,dno>
  - Which is better?
  - What if we consider the second query?

```
SELECT  E.dno, COUNT (*)
FROM  Emp E
WHERE  E.age=30
GROUP BY E.dno
```

```
SELECT  E.dno, COUNT (*)
FROM  Emp E
WHERE  E.age>30
GROUP BY E.dno
```

Index-Only Plans (Contd.)

- Index-only plans can also be found for queries involving more than one table;

<E.dno>

```
SELECT  D.mgr
FROM  Dept D, Emp E
WHERE  D.dno=E.dno
```

<E.dno,E.eid>

```
SELECT  D.mgr, E.eid
FROM  Dept D, Emp E
WHERE  D.dno=E.dno
```

Summary

- Many alternative file organizations exist, each appropriate in some situation.
- If selection queries are frequent, sorting the file or building an *index* is important.
  - Hash-based indexes only good for equality search.
  - Sorted files and tree-based indexes best for range search; also good for equality search. (Files rarely kept sorted in practice; B+ tree index is better.)
- Index is a collection of data entries plus a way to quickly find entries with given key values.

Summary (Contd.)

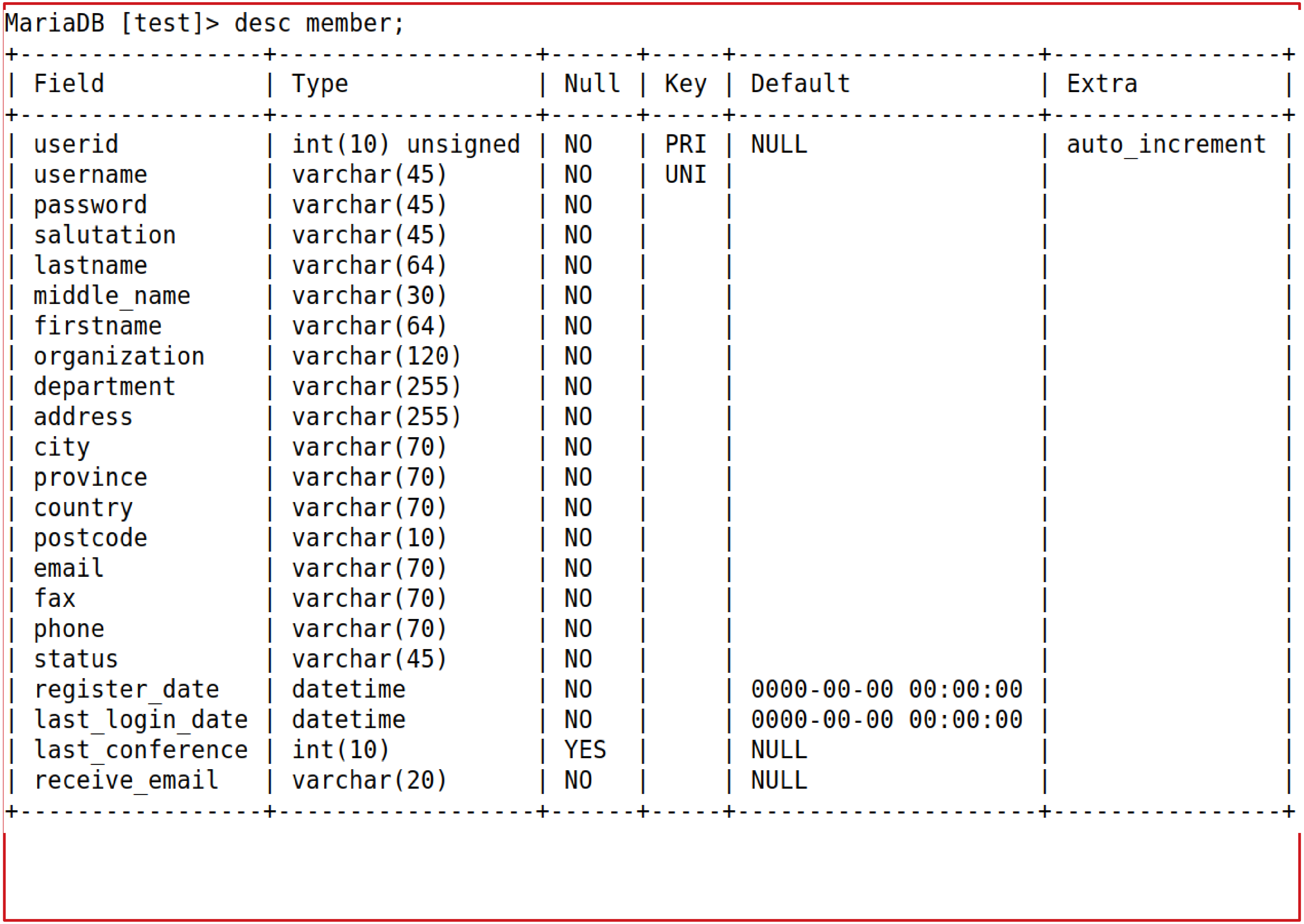
- Data entries can be actual data records, <key, rid> pairs, or <key, rid-list> pairs.
  - Choice orthogonal to *indexing technique* used to locate data entries with a given key value.
- Can have several indexes on a given file of data records, each with a different search key.
- Indexes can be classified as clustered vs. unclustered, primary vs. secondary, and dense vs. sparse. Differences have important consequences for utility/performance.

Summary (Contd.)

- Understanding the nature of the *workload* for the application, and the performance goals, is essential to developing a good design.
  - What are the important queries and updates? What attributes/relations are involved?
- Indexes must be chosen to speed up important queries (and perhaps some updates!).
  - Index maintenance overhead on updates to key fields.
  - Choose indexes that can help many queries, if possible.
  - Build indexes to support index-only strategies.
  - Clustering is an important decision; only one index on a given relation can be clustered!
  - Order of fields in composite index key can be important.

Database Index
&
Performance Optimization

Bipin C. DESAI



```
MariaDB [test]> desc member;
+-----------------+------------------+------+-----+---------------------+----------------+
| Field           | Type             | Null | Key | Default             | Extra          |
+-----------------+------------------+------+-----+---------------------+----------------+
| userid          | int(10) unsigned | NO   | PRI | NULL                | auto_increment |
| username        | varchar(45)      | NO   | UNI |                     |                |
| password        | varchar(45)      | NO   |     |                     |                |
| salutation      | varchar(45)      | NO   |     |                     |                |
| lastname        | varchar(64)      | NO   |     |                     |                |
| middle_name     | varchar(30)      | NO   |     |                     |                |
| firstname       | varchar(64)      | NO   |     |                     |                |
| organization    | varchar(120)     | NO   |     |                     |                |
| department      | varchar(255)     | NO   |     |                     |                |
| address         | varchar(255)     | NO   |     |                     |                |
| city            | varchar(70)      | NO   |     |                     |                |
| province        | varchar(70)      | NO   |     |                     |                |
| country         | varchar(70)      | NO   |     |                     |                |
| postcode        | varchar(10)      | NO   |     |                     |                |
| email           | varchar(70)      | NO   |     |                     |                |
| fax             | varchar(70)      | NO   |     |                     |                |
| phone           | varchar(70)      | NO   |     |                     |                |
| status          | varchar(45)      | NO   |     |                     |                |
| register_date   | datetime         | NO   |     | 0000-00-00 00:00:00 |                |
| last_login_date | datetime         | NO   |     | 0000-00-00 00:00:00 |                |
| last_conference | int(10)          | YES  |     | NULL                |                |
| receive_email   | varchar(20)      | NO   |     | NULL                |                |
+-----------------+------------------+------+-----+---------------------+----------------+
```

Create INDEX

```
MariaDB [test]> create index cntr_indx on member(country);
Query OK, 0 rows affected (0.061 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Creating an index on multiple columns

MariaDB/MySQL allows composite(multi-column) index
(up to 16 columns)

Usually 2 or 3 columns are sufficient

```
CREATE INDEX index_name ON TableName (Col1, COL2, COl3);
```

Drop INDEX

Drop index syntax

```
ALTER TABLE table_name DROP INDEX index_name;
```

Rename index syntax

```
ALTER TABLE table_name RENAME INDEX index_name
    TO new_index_name;
```

Show indexes syntax

```
SHOW INDEX FROM tableName;
```

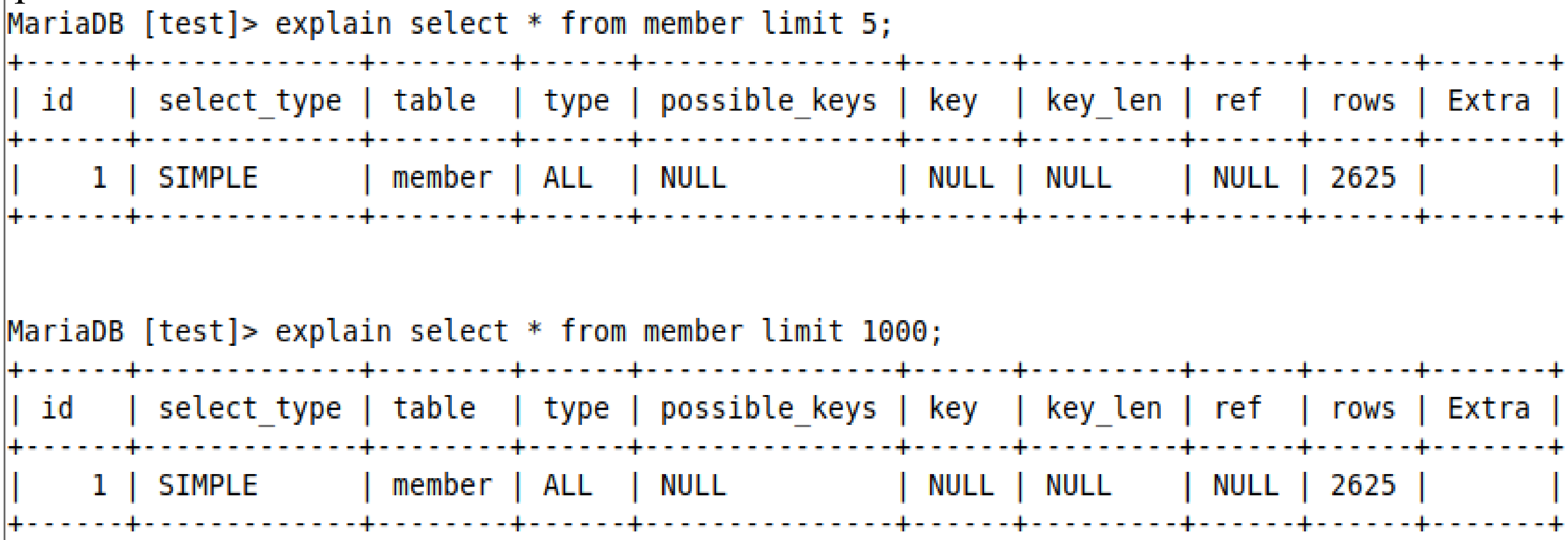
EXPLAIN

One can use EXPLAIN to see how the DB executes a DML statement

DML statements are:

SELECT, DELETE, INSERT, REPLACE, and UPDATE statements.

EXPLAIN gives execution plan information from the built-in DB optimizer



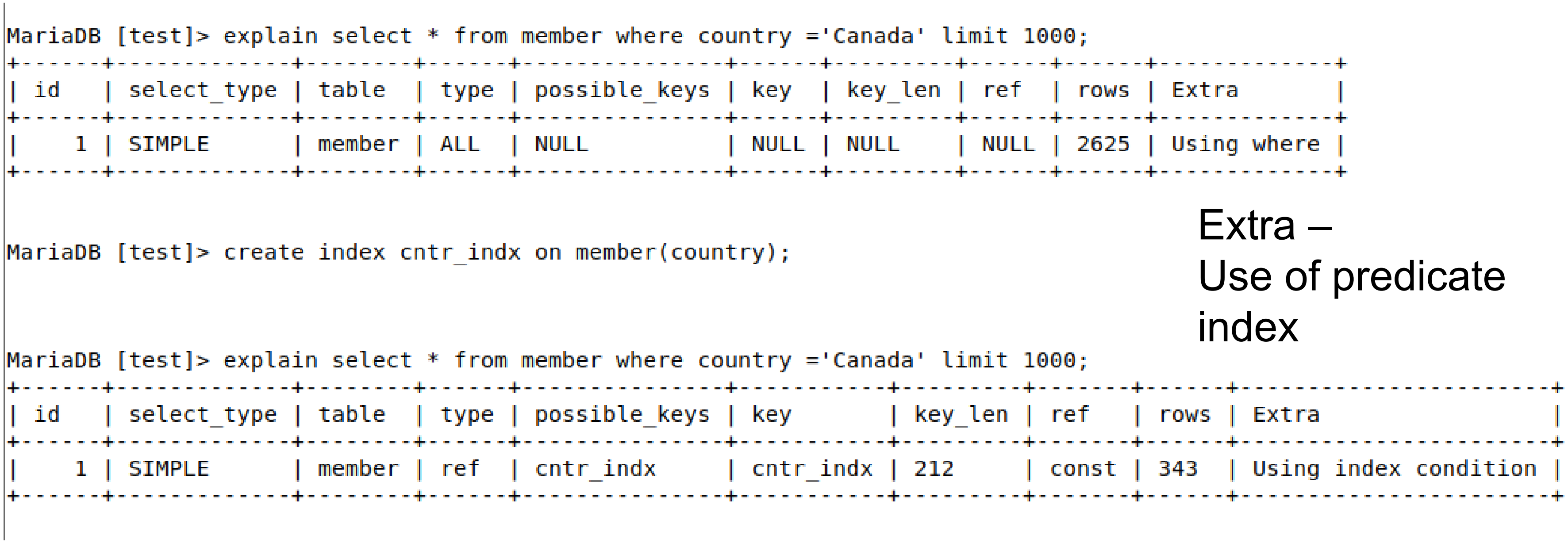
```
MariaDB [test]> explain select * from member limit 5;
+------+-------------+--------+------+---------------+------+---------+------+------+-------+
| id   | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra |
+------+-------------+--------+------+---------------+------+---------+------+------+-------+
|    1 | SIMPLE      | member | ALL  | NULL          | NULL | NULL    | NULL | 2625 |       |
+------+-------------+--------+------+---------------+------+---------+------+------+-------+


MariaDB [test]> explain select * from member limit 1000;
+------+-------------+--------+------+---------------+------+---------+------+------+-------+
| id   | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra |
+------+-------------+--------+------+---------------+------+---------+------+------+-------+
|    1 | SIMPLE      | member | ALL  | NULL          | NULL | NULL    | NULL | 2625 |       |
+------+-------------+--------+------+---------------+------+---------+------+------+-------+
```

The null for the “possible_keys” and “key” above are both NULL

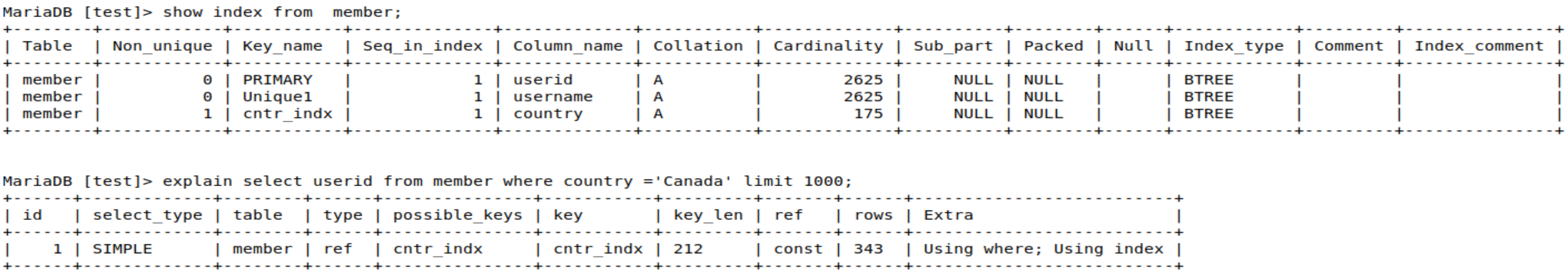
This indicates that the DB does not have an index it can use

The DB will access 2625 rows to generate the result

Extra –
Use of predicate index



```
MariaDB [test]> explain select * from member where country ='Canada' limit 1000;
+------+-------------+--------+------+---------------+------+---------+------+------+-------------+
| id   | select_type | table  | type | possible_keys | key  | key_len | ref  | rows | Extra       |
+------+-------------+--------+------+---------------+------+---------+------+------+-------------+
|    1 | SIMPLE      | member | ALL  | NULL          | NULL | NULL    | NULL | 2625 | Using where |
+------+-------------+--------+------+---------------+------+---------+------+------+-------------+

MariaDB [test]> create index cntr_indx on member(country);

MariaDB [test]> explain select * from member where country ='Canada' limit 1000;
+------+-------------+--------+------+---------------+-----------+---------+-------+------+-----------------------+
| id   | select_type | table  | type | possible_keys | key       | key_len | ref   | rows | Extra                 |
+------+-------------+--------+------+---------------+-----------+---------+-------+------+-----------------------+
|    1 | SIMPLE      | member | ref  | cntr_indx     | cntr_indx | 212     | const | 343  | Using index condition |
+------+-------------+--------+------+---------------+-----------+---------+-------+------+-----------------------+
```



```
MariaDB [test]> show index from  member;
+--------+------------+-----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table  | Non_unique | Key_name  | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+--------+------------+-----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| member |          0 | PRIMARY   |            1 | userid      | A         |        2625 |     NULL | NULL   |      | BTREE      |         |               |
| member |          0 | Unique1   |            1 | username    | A         |        2625 |     NULL | NULL   |      | BTREE      |         |               |
| member |          1 | cntr_indx |            1 | country     | A         |         175 |     NULL | NULL   |      | BTREE      |         |               |
+--------+------------+-----------+--------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+

MariaDB [test]> explain select userid from member where country ='Canada' limit 1000;
+------+-------------+--------+------+---------------+-----------+---------+-------+------+--------------------------+
| id   | select_type | table  | type | possible_keys | key       | key_len | ref   | rows | Extra                    |
+------+-------------+--------+------+---------------+-----------+---------+-------+------+--------------------------+
|    1 | SIMPLE      | member | ref  | cntr_indx     | cntr_indx | 212     | const | 343  | Using where; Using index |
+------+-------------+--------+------+---------------+-----------+---------+-------+------+--------------------------+
```

Course Notes

Files and Databases

Bipin C. DESAI

Limit of Liability/Disclaimer of Warranty:

The authors and the publishers have taken care to prepare this book. However, there is no warranty of the accuracy, completeness or presentation of the latest version/generation of any system discussed in this book. The reader must be aware of the fact that software systems often have multiple bugs and are not well thought out, and are usually suitable for limited situations and/or data combinations. Hence the user must be responsible for the appropriate application of any technique and use of any software or code examples.

Furthermore, there is no assurance whatsoever of the possible usefulness or commercialization of any programs, scripts and examples given in this book.

Any references given are based on their existence at the time of writing and the authors and the publishers do not endorse them or imply any usefulness of the information found therein. The reader must be aware that any web site cited may change, disappear or change their terms of service.

Published by: Electronic Publishing BytePress.com Inc.
Electronic - ISBN: 978-1-988392-16-5

SQL III - Relational Object Features

- Handling complex data and OO Concept
- Using structured data types and inheritance in SQL
- Object Identity (OID) and reference types in SQL
- Implementing Object features in relational DBMS
- Persistent Programming Languages
- Include object orientation and constructs to deal with added data types in RDBMS.
- Add complex types, including non-atomic values such as nested relations.
- Extend modeling features while retaining the declarative access to data
- Preserve compatibility with SQL

This is what we started with!

123	Smith	D1	P1	5	L1
			P2	30	L1
234	Ma	D2	P1	20	L1
			P3	10	L2
			P4	5	L3
345	Russo	D1	P1	35	L1

Example of a non-normal form (NNF) relation

123	Smith	D1	P1	5	L1
			P2	30	L1
234	Ma	D2	P1	20	L1
			P3	10	L2
			P4	5	L3
345	Russo	D1	P1	35	L1

A non-normal form relation →
Nested relation

- Abandon atomic attribute requirement by conceptually allowing nested relations — relations within relations
- Maintain the mathematical foundation of relational model
- Allow NNF i. e, non-normal form

Complex data

Example of a relation with complex data including: array, composite data, sets

Title	Authors array	Publication (Name, Vol, Date, pages)	Meeting	Subject set
Report of the Priorities Workshop	**[Caillau, Desai]**	(Computer Networks and ISDN Systems; Vol. 27-2,;November 1994; pp. 334-336)	WWW-I	{Web, searching}
Three Paradoxes of Big data	[Richards, King]	(Stanford, CA,;;Sept, 2013,;pp102-1050)	Making End Meet	{Big Data, security, privacy]

Decomposition of complex data into

4NF decomposition!

A 4NF relation does not have any multivalued dependency of the form $X \rightarrow\rightarrow Y$

Report of the Priorities Workshop →→ {Caillau, Desai}

Title	Authors
Report of the Priorities Workshop	Caillau,
Report of the Priorities Workshop	Desai
Three Paradoxes of Big data	Richards
Three Paradoxes of Big data	King

Decomposition of complex data into 4NF decomposition (contd.)!

Title	Name	Vol,	Date	pages
Report of the Priorities Workshop	Computer Networks and ISDN Systems	Vol. 27-2	Nov. 1994	pp. 334-336
Three Paradoxes of Big data	Stanford, CA		Sept, 2013,	pp102-1050

Decomposition of complex data into 4NF decomposition (contd.)!

Title	Meeting
Report of the Priorities Workshop	WWW-I
Three Paradoxes of Big data	Making End Meet

Title	Subject
Report of the Priorities Workshop	Web
Report of the Priorities Workshop	searching
Three Paradoxes of Big data	Big Data
Three Paradoxes of Big data	security
Three Paradoxes of Big data	privacy

Postgresql

Ingres was one of first relational ‘open source” relational Database that was developed in the early 1970s at UoC, Berkley It gave rise to, among others, SysBase, Microsoft SQL server etc. Ingres was followed by Postgres and Postgresql

It is available for various versions of Linuses and other OS.

To install in

- fedora core: dnf -y install postgres*
- Debian, Ubuntu: apt install postgresql postgresql-contrib

PostgreSQL is sometimes called an "object-relational database" because it supports table inheritance.

Most of the ORDBMS features were removed from the Postgres and became PostgreSQL.

Once PostgresSQL is installed the database is initialized using:

```
initdb -D /path-to/postgres/data
```

One can now start the database server using:

```
pg_ctl -D /path-to/postgres/data -l logfile start
```

Once the server is running, a database could be created using:

```
createdb testdb
```

Postgresql shell

The shell can be started using the command: psql

To exit from psql use: \q or CTRL-D

To get a list of commands use: \?

To use a particular database use: \c nameofDB;

To list all the databases use; \l;

To stop Postgresql server use the command

```
pg_ctl -D /path-to/postgres/data -l logfile stop
```

Postgres objects

```
CREATE TABLE publication (
    title            text primary key,
    authors  text[],
    meeting        text,
    publication text[][],
    topics text []
);
```

1 dimensional array

2 dimensional array

```
postgres=# \d+ publication;
                        Table "public.publication"
   Column    |  Type  | Modifiers | Storage  |Stats target|Description
-------------+--------+-----------+----------+------------+-----------
 title       | text   | not null  | extended |            |
 authors     | text[] |           | extended |            |
 meeting     | text   |           | extended |            |
 publication | text[] |           | extended |            |
 topics      | text[] |           | extended |            |
Indexes:
    "publication_pkey" PRIMARY KEY, btree (title)

Has OIDs: no
```

```
insert into publication values(
'Report of the Priorities Workshop',
ARRAY['Caillau', 'Desai'],
'WWW-I',
ARRAY[['event', 'Computer Networks and ISDN Systems'],
['volume','27-2'], ['year', 'November 1994'], ['pages','pp. 334-336']],
ARRAY['Web', 'searching']);
```

```
postgres=# select * from publication;
Report of the Priorities Workshop | {Caillau,Desai} | WWW-I    |
{{event,"Computer Networks and ISDN Systems"},{volume,27-2},
{year,"November 1994"},{pages,"pp. 334-336"}} | {Web,searching}
(1 row)
```

Use the unnest() function to convert array to set of rows:

```
SELECT *
 FROM (
          SELECT *, unnest(authors) allauthors
              FROM publication ) x
             WHERE allauthors LIKE 'Cail%';
```

```
title        |    authors    | meeting |  publication  |   topics   | allauthors
Report of the Priorities Workshop | {Caillau,Desai} | WWW-I   |
{{event,"Computer Networks and ISDN Systems"},{volume,27-2},
{year,"November 1994"},{pages,"pp.334-336"}} |
{Web,searching} | Caillau
```

SQL 1999 extended to support complex types:

- Collection and large object types
  - Nested relations are an example of collection types
- Structured types: arbitrary hierarchies and composite attributes
- Inheritance
- Object orientation: object identifiers and references

SQL 1999 is yet to be fully implemented in most DBMS (2014)

- Examples of OODBMS are:
- ObjectStore, Objectdatabase++, Objectivity/DB, etc.

Some RDBMS have introduced some object features

OODBMS feature including using object oriented language to manipulate database objects along with the others of RDBMS

(ACID, Query language, Recovery)

Oracle: Creating type (class)and a nested table[1]

```
CREATE OR REPLACE TYPE person_typ AS OBJECT (
idno NUMBER,
name VARCHAR2(30),
phone VARCHAR2(20),
MAP MEMBER FUNCTION get_idno RETURN NUMBER,
MEMBER PROCEDURE display_details
( SELF IN OUT NOCOPY person_typ ) );
/
Type created.
```

The SELF parameter denotes the object instance currently invoking the method. NOCOPY allows passing the argument by reference (i.e., not copying the argument to the method)

[1] Based on old Oracle documents

Member Methods for Comparing Objects

An object type, with multiple attributes of various data types, has no predefined axis of comparison.

Methods should be specified to compare & order object type variables

The option is to define an map method or an order method for comparing objects, but not both.

Map Methods

Map methods return values that can be used for comparing and sorting.

Return values can be any built-in data types(except LOBs and BFILEs)

Order Methods

An order method directly compares values for two particular objects.

```
SQL> desc person_typ;
 Name      Null?   Type
 -------- -------- --------------
 IDNO               NUMBER
 NAME               VARCHAR2(30)
 PHONE              VARCHAR2(20)

METHOD
------
 MAP MEMBER FUNCTION GET_IDNO RETURNS NUMBER
 MEMBER PROCEDURE DISPLAY_DETAILS
```

```
CREATE OR REPLACE TYPE BODY person_typ AS
MAP MEMBER FUNCTION get_idno RETURN NUMBER IS
BEGIN
RETURN idno;
END;
MEMBER PROCEDURE display_details
      ( SELF IN OUT NOCOPY person_typ ) IS
BEGIN   -- use the put_line procedure of the DBMS_OUTPUT
        -- package to display details
DBMS_OUTPUT.put_line(TO_CHAR(idno)|| ' - '|| name|| ' - ' || phone);
END;
END;
/
Type body created.
```

```
CREATE OR REPLACE TYPE
people_typ AS TABLE OF
 person_typ; -- nested table type
/
```

Creating an Instance of a VARRAY or Nested Table

To create an instance of a collection type by calling the constructor method of the type.

The constructor method is the name of the type.

The elements of the collection is a comma-delimited list of arguments to the method, for example.

```
person_typ(1, 'John Smith', '1-650-555-0135')
```

- Create a table that contains an instance of the nested table type people_typ, named people_column,

-use the constructor method in a SQL statement to insert values into people_typ.

Example: Using the Constructor Method to Insert Values into a Nested Tab

```
CREATE TABLE people_tab (
group_no NUMBER,
people_column people_typ ) -- an instance of nested table
NESTED TABLE people_column STORE AS people_column_nt
/
Table created.
```

```
INSERT INTO people_tab VALUES ( 100,
people_typ( person_typ(1, 'John Smith', '1-650-555-0135'),
person_typ(2, 'Diane Smith', NULL))) ;
1 row created.
```

Create a department_persons Table Using the DEFAULT Clause

```
CREATE TABLE department_persons (
dept_no NUMBER PRIMARY KEY,
dept_name CHAR(20),
dept_mgr person_typ DEFAULT person_typ(10,'John Doe',NULL),
dept_emps people_typ DEFAULT people_typ() )
NESTED TABLE dept_emps STORE AS dept_emps_tab;
Table created.
```

instance of nested table type

```
SQL> desc department_persons;
 Name        Null?     Type
 ---------- --------- ----------------
 DEPT_NO     NOT NULL NUMBER
 DEPT_NAME            CHAR(20)
 DEPT_MGR              PERSON_TYP
 DEPT_EMPS             PEOPLE_TYP
```

```
INSERT INTO department_persons VALUES
(101, 'Physical Sciences',  person_typ(65,'Vrinda Mills', '1-650-555-0125'),
people_typ( person_typ(1, 'John Smith', '1-650-555-0135'),
person_typ(2, 'Diane Smith', NULL) ) );
INSERT INTO department_persons VALUES
( 104, 'Life Sciences', person_typ(70,'James Hall', '1-415-555-0101'),
people_typ() ) -- an empty people_typ table
-- Note that people_typ() is a literal invocation of the constructor
-- method for an empty people_typ nested table.
/
```

select * from department_persons;

```
DEPT_NO DEPT_NAME
---------- --------------------
DEPT_MGR(IDNO, NAME, PHONE)
-------------------------------------------------
DEPT_EMPS(IDNO, NAME, PHONE)
-------------------------------------------------
       101 Physical Sciences
PERSON_TYP(65, 'Vrinda Mills', '1-650-555-0125')
PEOPLE_TYP(PERSON_TYP(1, 'John Smith', '1-650-555-0135'),
PERSON_TYP(2, 'Diane Smith', NULL))
       104 Life Sciences
PERSON_TYP(70, 'James Hall', '1-415-555-0101')
   DEPT_NO DEPT_NAME
---------- --------------------
DEPT_MGR(IDNO, NAME, PHONE)
------------------------------------------------
DEPT_EMPS(IDNO, NAME, PHONE)
------------------------------------------------
PEOPLE_TYP()
```

Nesting Results of Collection Queries

```
SELECT d.dept_emps
FROM department_persons d;
```

Example shows the query retrieving the nested collection of employees from the department_persons table

The column dept_emps is a nested table collection of person_typ type.

The dept_emps collection column appears in the SELECT list as an Ordinary scalar column.

Querying a collection column in the SELECT list this way nests the elements of the collection in the result row that the collection is associated with.

```
DEPT_EMPS(IDNO, NAME, PHONE)
-------------------------------------------------------------
PEOPLE_TYP(PERSON_TYP(1, 'John Smith', '1-650-555-0135'),
PERSON_TYP(2, 'Diane Smith', NULL))
PEOPLE_TYP()
```

Unnesting Results of Collection Queries

Not all tools or applications can deal with results in a nested format.

To view collection data using tools that require a conventional format, one must un-nest, the collection attribute of a row into one or more relational rows using a TABLE expression

TABLE expressions enable you to query a collection in the FROM clause as a table.

In effect, you join the nested table with the row that contains the nested table.

TABLE expressions can be used to query any collection value expression, including transient values such as variables and parameters.

Example Un-nesting Results of Collection Queries

```
SELECT e.*
FROM department_persons d, TABLE(d.dept_emps) e;

      IDNO NAME                           PHONE
---------- ------------------------------ --------------------
         1 John Smith                     1-650-555-0135
         2 Diane Smith
```

Creating and Populating Simple Nested Tables

```
CREATE TABLE students (
graduation DATE,
math_majors people_typ, -- nested tables (empty)
chem_majors people_typ,
physics_majors people_typ)
NESTED TABLE math_majors STORE AS math_majors_nt
                          -- storage tables
NESTED TABLE chem_majors STORE AS chem_majors_nt
NESTED TABLE physics_majors STORE AS physics_majors_nt;

Table created.
```

```
SQL> desc students;
 Name          Null?    Type
 ------------- -------- ---------------
 GRADUATION             DATE
 MATH_MAJORS            PEOPLE_TYP
 CHEM_MAJORS            PEOPLE_TYP
 PHYSICS_MAJORS              PEOPLE_TYP
```

The NESTED TABLE..STORE AS clause specifies storage names for nested tables.

Elements of a nested table are actually stored in a separate storage table.

Storage names -used to create an index on a nested table.

```
CREATE INDEX math_idno_idx ON math_majors_nt(idno);
CREATE INDEX chem_idno_idx ON chem_majors_nt(idno);
CREATE INDEX physics_idno_idx ON physics_majors_nt(idno);
```

```
INSERT INTO students (graduation) VALUES
 ('01-JUN-03');

SQL> select * from students;
GRADUATION
------------------
MATH_MAJORS(IDNO, NAME, PHONE)
-----------------------------------
CHEM_MAJORS(IDNO, NAME, PHONE)
-----------------------------------
PHYSICS_MAJORS(IDNO, NAME, PHONE)
-----------------------------------
01-JUN-03
```

```
UPDATE students
SET math_majors =
people_typ (person_typ(12, 'Bob Jones', '650-555-0130'),
person_typ(31, 'Sarah Chen', '415-555-0120'),
person_typ(45, 'Chris Woods', '415-555-0124')),
chem_majors =
people_typ (person_typ(51, 'Joe Lane', '650-555-0140'),
person_typ(31, 'Sarah Chen', '415-555-0120'),
person_typ(52, 'Kim Patel', '650-555-0135')),
physics_majors =
people_typ (person_typ(12, 'Bob Jones', '650-555-0130'),
person_typ(45, 'Chris Woods', '415-555-0124'))
WHERE graduation = '01-JUN-03';
```

```
SQL> select * from students;
```

```
GRADUATION
------------------
MATH_MAJORS(IDNO, NAME, PHONE)
------------------------------------
CHEM_MAJORS(IDNO, NAME, PHONE)
------------------------------------
PHYSICS_MAJORS(IDNO, NAME, PHONE)
------------------------------------
01-JUN-03
PEOPLE_TYP(PERSON_TYP(12, 'Bob Jones', '650-555-0130'),
PERSON_TYP(31, 'Sarah Chen', '415-555-0120'),
PERSON_TYP(45, 'Chris Woods', '415-555-0124'))

PEOPLE_TYP(PERSON_TYP(51, 'Joe Lane', '650-555-0140'),
PERSON_TYP(31, 'Sarah Chen', '415-555-0120'),
PERSON_TYP(52, 'Kim Patel', '650-555-0135'))

PEOPLE_TYP(PERSON_TYP(12, 'Bob Jones', '650-555-0130'),
PERSON_TYP(45, 'Chris Woods', '415-555-0124'))
```

```
select owner, object_name, object_type
  from ALL_OBJECTS
 where object_type = 'TYPE'and owner='BCDESAI';

select owner, object_name, object_type
  from ALL_OBJECTS
 where object_type = 'TABLE' and owner='BCDESAI';
```

- Following are some slides from

Database System Concepts, 6th Ed.

©Silberschatz, Korth and Sudarshan
corrected for
Oracle by BCD

Structured Types and Inheritance in SQL

Structured types (a.k.a. **user-defined types**) can be declared & used in SQL

```
create type Name as object
        (firstname        varchar(20),
         lastname         varchar(20))
              final
     create type Address as object
        (street        varchar(20),
         city    varchar(20),
         zipcode       varchar(20))
              not final
```

– Note: **final** and **not final** indicate whether subtypes can be created

```
SQL> desc Name;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 FIRSTNAME                                          VARCHAR2(20)
 LASTNAME                                           VARCHAR2(20)


SQL> desc address;
 address is NOT FINAL
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 STREET                                         VARCHAR2(20)
 CITY                                       VARCHAR2(20)
 ZIPCODE                                         VARCHAR2(20)
```

Structured Types and Inheritance in SQL

- Structured types can be used to create tables with composite attributes

```
create table person (
                name   Name,
                address         Address,
                dateOfBirth date)
```

- Dot notation used to reference components: *name.firstname*

```
SQL> desc person;
 Name                                      Null?    Type
 ----------------------------------------- -------- --------------------
 NAME                                               NAME
 ADDRESS                                            ADDRESS
 DATEOFBIRTH                                        DATE
```

Structured Types (cont.)

- **User-defined row types**

  create type *PersonType* **as object (**
  name Name,
  address Address,
  dateOfBirth **date)**
  not final

```
Warning: Type created with compilation errors.
SQL> show errors
Errors for TYPE PERSONTYPE:
LINE/COL ERROR
-------- -----------------------------------------------------------------
0/0      PL/SQL: Compilation unit analysis terminated
2/6      PLS-00320: the declaration of the type of this expression is
         incomplete or malformed
```

- Once a type is created, we can create one or more tables using our (user-defined) type

  create table *customer* **of** *~~Customer~~PersonType*

Structured Types (cont.)

- Alternative method which uses **unnamed row types**.

```
create table person_r(
                  name    row(firstname  varchar(20),
                          lastname  varchar(20)),
                  address  row(street     varchar(20),
                              city        varchar(20),
                             zipcode  varchar(20)),
                  dateOfBirth date)
```

row is not supported in oracle!!!

Methods

In ORDMS, we can add a method declaration with a structured type.

```
        method ageOnDate (onDate date)
                returns interval year
```

Method body is given separately.

```
    create instance method ageOnDate (onDate date)
                    returns interval year
                    for CustomerType
    begin
                    return onDate - self.dateOfBirth;
    end
```

We can now find the age of each customer:

```
    select name.lastname, ageOnDate (current_date)
    from customer
```

Constructor Functions

Constructor functions are used to create values of structured types

```
create function Name(firstname varchar(20), lastname varchar(20))
returns Name
begin
   set self.firstname = firstname;
   set self.lastname = lastname;
end
```

To create a value of type *Name,* we use

```
  new Name(‘John’, ‘Smith’)
```

Normally used in insert statements

```
insert into Person values
    (new Name(‘John’, ‘Smith),
     new Address(’20 Main St’, ‘New York’, ‘11001’),
     date ‘1960-8-22’);
```

Type Inheritance

Suppose that we have the following type definition for people:

```
      create type Person
         (name varchar(20),
          address varchar(20))
```

Using inheritance to define the student and teacher types

```
   create type Student
    under Person
    (degree      varchar(20),
     department  varchar(20))
   create type Teacher
    under Person
    (salary       integer,
     department  varchar(20))
```

Subtypes can redefine methods by using **overriding method** in place of **method** in the method declaration

Multiple Type Inheritance

SQL:1999 and SQL:2003 do not support multiple inheritance

If our type system supports multiple inheritance, we can define a type for teaching assistant as follows:

```
        create type Teaching Assistant
        under Student, Teacher
```

To avoid a conflict between the two occurrences of *department* we can rename them

```
           create type Teaching Assistant
       under
        Student  with (department as student_dept ),
       Teacher  with (department as teacher_dept )
```

Each value must have a **most-specific type**

Table Inheritance

Tables created from subtypes can further be specified as **subtables**

E.g. **create table** *people* **of** *Person;*

create table *students* **of** *Student* **under** *people;*

create table *teachers* **of** *Teacher* **under** *people;*

Tuples added to a subtable are automatically visible to queries on the super-table

E.g. query on *people* also sees *students* and *teacher*s.

Similarly updates/deletes on *people* also result in updates/deletes on subtables

To override this behaviour, use "**only** *people"* in query

Conceptually, multiple inheritance is possible with tables

e.g. *teaching_assistants* under *students* and *teachers*

Not supported in SQL currently*:* So we cannot create a person (tuple in *people*) who is both a student and a teacher

JSON Objects

SGML ⇒ XML ⇒ JSON

JSON Java Script Object Notation

```
MariaDB [test]> select JSON_OBJECT
( 'name', 'Don Duck',
'IQ', 'Calm Genius') as top_duck;
+-------------------------------------------+
| top_duck                                  |
+-------------------------------------------+
| {"name": "Don Duck", "IQ": "Calm Genius"} |
+-------------------------------------------+
1 row in set (0.001 sec)
```

NoSQL

Brewer’s (CAP) Theorem

There are three core systemic requirements that exist in a special inter-relationship when it comes to designing and deploying applications in a distributed environment

The three requirements are:

Consistency,

Availability and

Partition Tolerance

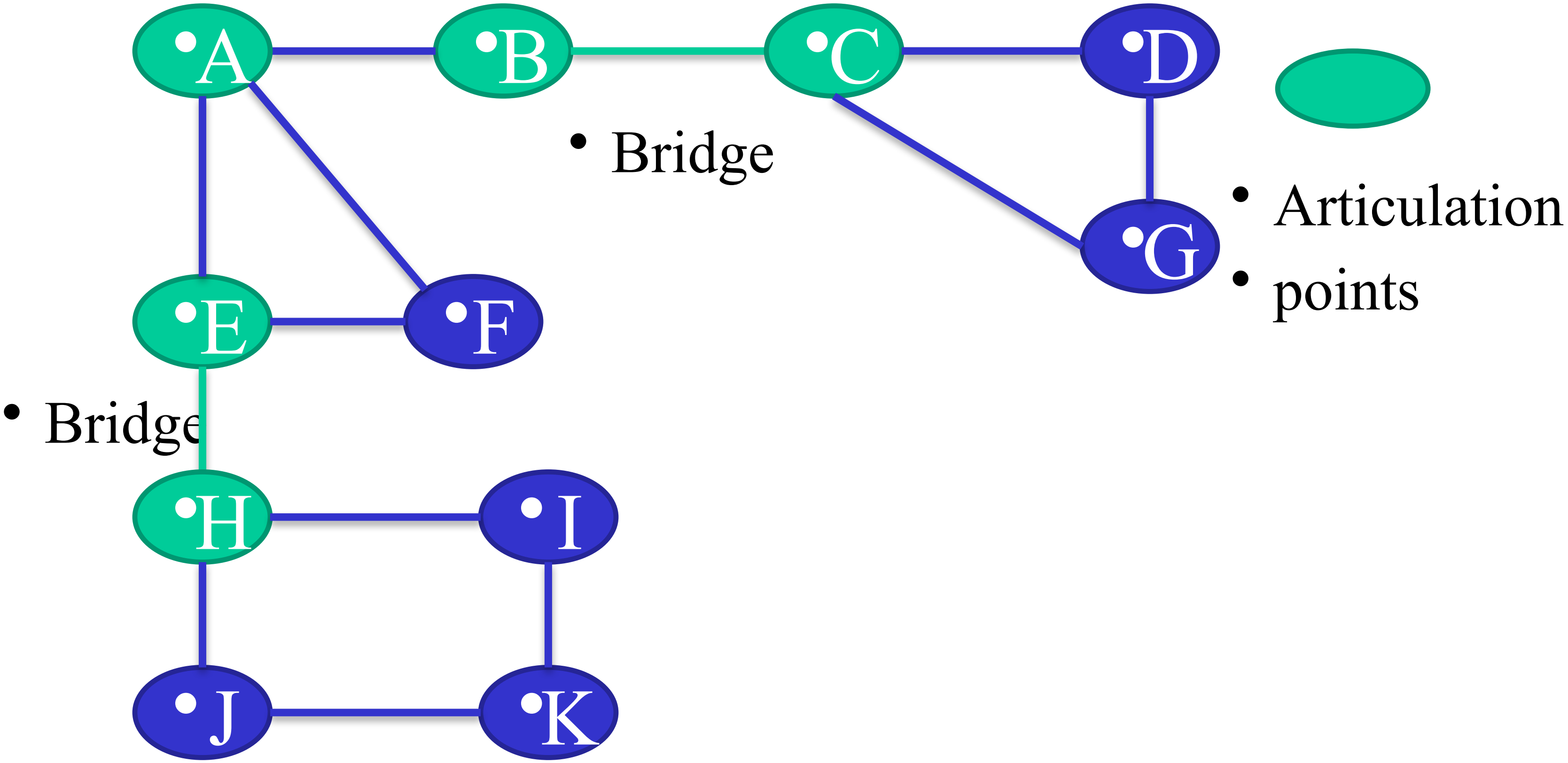
Compare these with the ACID property that is the traditional requirement

Atomicity – all or nothing

Consistency - the data goes from one consistent state to another

Isolation – a transaction is guaranteed to run as if it was the only one

Durability – any changes made by a transaction are persistent



A distributed system with articulation points (cut vertex) removing which disconnects the graph and bridges(connects subgraphs)

Brewer’s (CAP) Theorem

Consistency: A constraint of distributed systems that multiple values for the same piece of data are not allowed. **Atomicity guarantees that all changes made by a transaction are made or there would be no changes**

Availability: Availability means that a service is available. Sites must not to go down at busy periods just because they are busy.

Partition Tolerance: A partition happens when, say, a bridge fails or an articulation node goes down

This causes the network to be partitioned.

Temporary partitions are a possible and critical systems should be tolerant to such events

Dealing with CAP – only two could be guaranteed!

Drop Partition Tolerance

Run on one system or have bullet proof distributed system

(not possible)

Drop Availability Tolerance

Economically and political downside.

Drop Consistency Tolerance :

This is the obvious choice in most cases.

Easy to deal with – the masses will not know!!!

NoSQL

New database applications and new databases – non-relational

Abandon the ACID property – substitute performance, scalability etc.

Group most often required data items together

– abandon normalization and the relational approach and hence eliminate joins

Cluster friendly- allow use of multitude of cheap servers

– distributed and partitioned

- No fixed schema (not really!)

NoSQL

Category of “model” and some implementations

Column family: BigTable(Google),**Cassandra**, Druid, Hadoop/HBase

Unique keys point to multiple columns.

The columns are arranged by column family.

Document: Apache CouchDB, Couchbase, MongoDB

Lotus Notes and are similar to key-value stores for semi-structured data

The semi-structured documents are stored in JSON like formats.

Key-value: Dynamo(Amazon), FoundationDB, MemcacheDB, Redis

A unique key with pointers to items of data: to implement.

inefficient when accessing small portion of data

Graph: Allegro, Neo4J, InfiniteGraph, OrientDB

A graph theory based model is used

See: http://nosql-database.org/ for a list of NoSQL databases

Hadoop

Hadoop is a software approach to implements massively parallel computing. **http://hadoop.apache.org/**

Hadoop modules:

Hadoop Common: The common utilities that support the other Hadoop modules.

Hadoop Distributed File System (HDFS™):

A distributed file system that provides high-throughput access to application data.

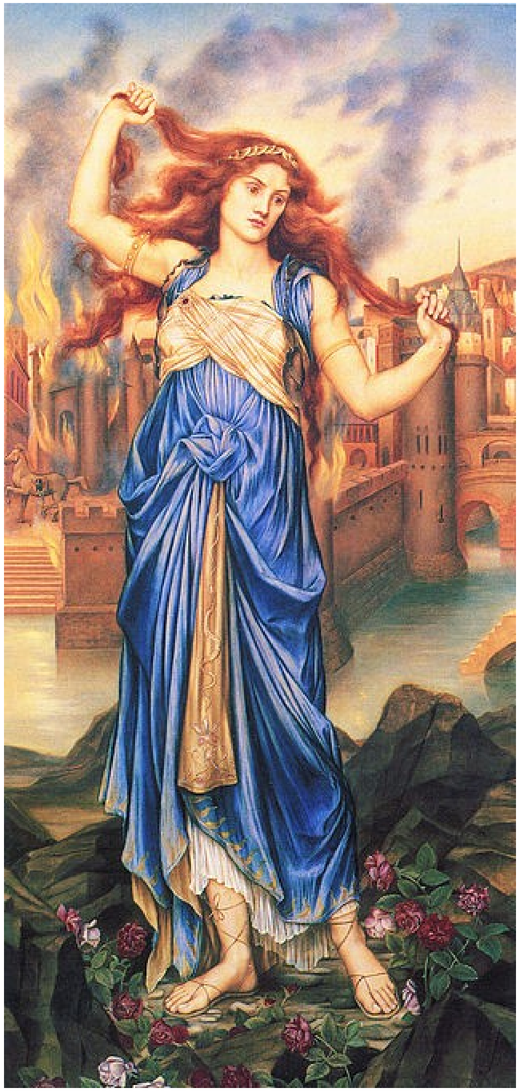
Hadoop YARN: A framework for job scheduling and cluster resource management.

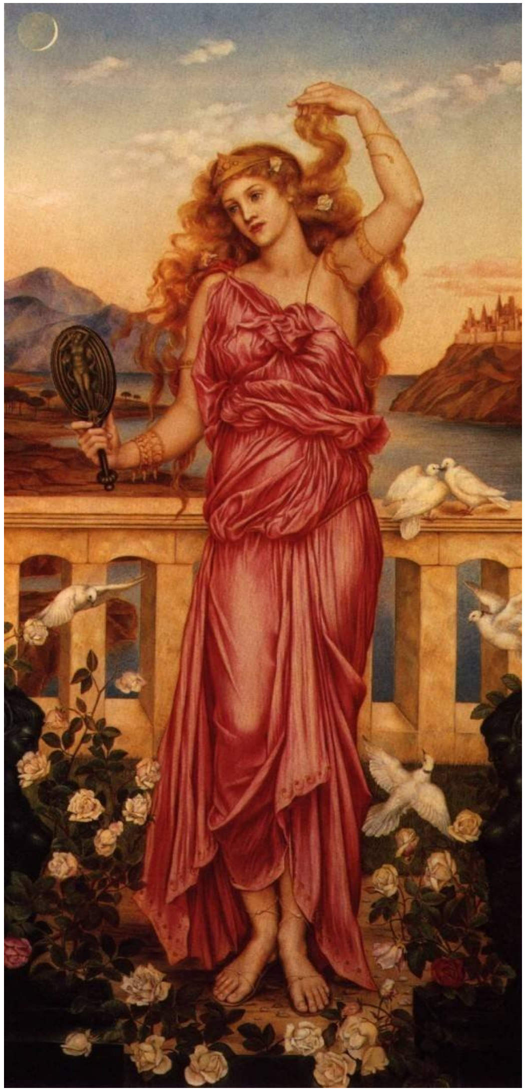
Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.

These modules provide feature that allow data to be spread across thousands of servers with little reduction in performance

- **Semi join** $\ltimes$ $\rtimes$
- A technique used to support join when a relational database is
- distributed over a number of nodes.
- Suppose table R is on node r and S is on node s and the common
- attribute of R and S is *C*.
- In semi-join of R $\ltimes$ S, we proceed as follows
- we send $\prod_C$ R from node r to node s
- at node we do a join of ($\prod_C$ R $\bowtie$ $\prod_C$ S)
- send the result to node r where we do
- R $\bowtie$ ($\prod_C$ R $\bowtie$ $\prod_C$ S)

- **MapReduce**, apply semi join like concept and distribute
- the computation over the nodes
- Answer to the processing needs of large amount of data





- ***Cassanda (1898)***

- *Helen of Troy* ***(1898)***

- **Paintings by Evelyn de Morgan**

Apache Cassandra

Cassandra is a NoSQL Column family implementation

Some of the strong points of Cassandra are:

Highly scalable and highly available with no single point of failure

NoSQL column family implementation

Very high write throughput and good read throughput

SQL-like query language (since 0.8) and support search through secondary indexes

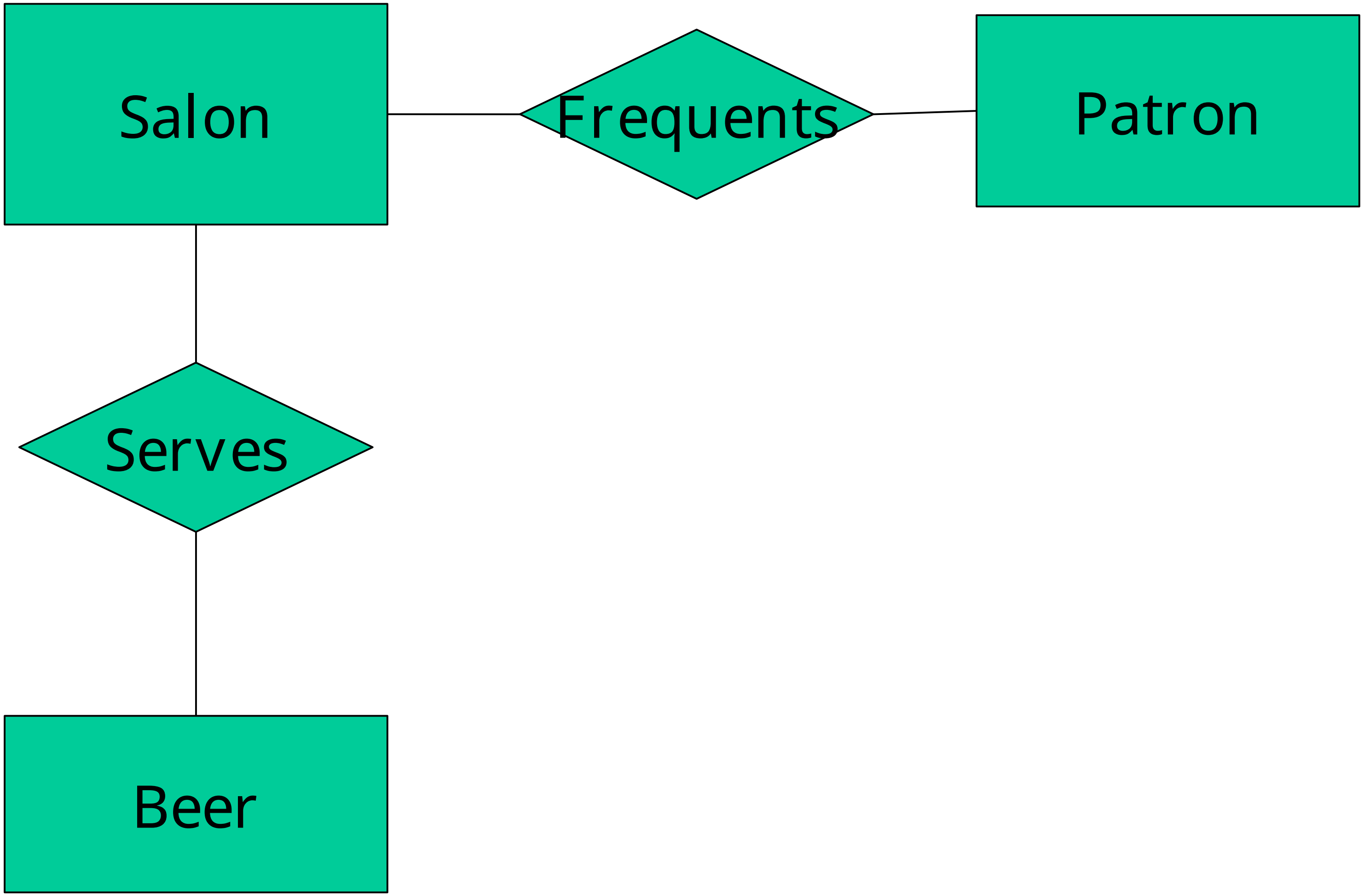
Tunable consistency and support for replication

Flexible schema

```
/usr/bin/cqlsh

Connected to Test Cluster at localhost:9160.

[cqlsh 4.1.1 | Cassandra 2.0.10 | CQL spec 3.1.1 | Thrift protocol 19.39.0]

Use HELP for help.

cqlsh> CREATE KEYSPACE testkeyspc  WITH REPLICATION = { 'class' :
    'SimpleStrategy', 'replication_factor' : 1 };

cqlsh> use testkeyspc;
```

Keyspace is a “database”

```
cqlsh:testkeyspc> describe keyspace;

CREATE KEYSPACE testkeyspc WITH replication = {

  'class': 'SimpleStrategy',

  'replication_factor': '1'

};

cqlsh:testkeyspc> >create table salon(

              ... sname text primary key,

              ... saddress text);
```



- Exercise: Install Cassandra on your desktop/laptop and implement this model