

CS3700

Introduction to Database Systems

Prof P Sreenivasa Kumar
Department of CSE, I I T Madras

Introduction to Database Systems CS3700

Objectives:

Introduce the learner to the fundamental ideas/ principles of relational databases and relational database management systems.

Next few slides:

A perspective of the “data” space

Prof P Sreenivasa Kumar, Department of CS&E, IITM.
2

Kinds of Data

- Various kinds of Data we encounter everyday:
 - Enterprise (business) data
 - Structured data
 - Documents / webpages – Text data
 - Objects – JSON; XML – (label, value) pairs data
 - Semi-structured data
 - Voice / music / image / video data
 - Unstructured data
- This course focuses on *structured data*

Prof P Sreenivasa Kumar, Department of CS&E, IITM.
3

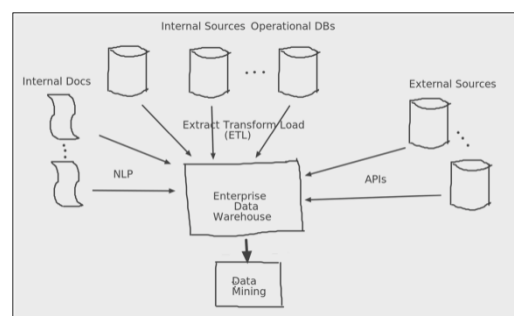
Data at Large Enterprises

- Large Enterprises (Businesses)
 - Need to deal with multiple types of data
 - E-commerce company
 - Product specs, images, demo videos, customer profiles ...
 - Transactions data
 - Multiples sources of data
 - Each division – its own data
 - External data – stock prices, media reports etc
 - Data Integration – a necessity
- Need multiple DBs and a Data Warehouse

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

4

Data Warehouses



Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

5

Data Mining vs Querying

- Data Mining – Knowledge Discovery in Databases
 - Discover hither-to unknown 'knowledge' in data
 - Knowledge – associations / clusters / classifications ...
 - Custom-designed algorithms
- DB Querying
 - Ask for details present in data or can be derived
 - Use logic-based expression languages – SQL/OQL etc
 - Ask for aggregated information
 - How many 'fiction' books got issued to 'CS' students in the last quarter ?

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

6

OLAP - OLTP

- OLAP (Online analytical processing) queries:
 - Ask for aggregated information from data
 - Needs summarization of large amount of data
 - How many 'cosmetics' products were sold in all the stores located in the 'southern' region of the country in the last 2 quarters (Big Basket or Amazon)
 - Specialized architecting of data is needed
 - Done on warehouse data using "Data Cubes"
- OLTP(Online transaction processing)
 - Process a transaction (sale / credit card charge / reservation) quickly – concurrency needs attention
 - An important functionality of DB servers

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

7

Text data - IR Systems

- IR – Information Retrieval Systems
 - Typified by Search Engines – Web/Enterprise
 - Query – list of keywords – (avg : 2.4 words/query)
 - Intention is imprecise
 - Relational queries are very precise (to see later)
 - Results – a ranked list(s) of documents
 - Ordered by relevance to the Query
 - Hugely successful
 - Modern IR systems – not just a repository of docs!
 - Needs a separate course

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

8

OO Languages and Databases

- Object-Oriented (OO) programming
 - Deeply structured objects – nesting is common
 - Rectangle – set of 4 point objects; Point – a pair
 - Objects are not 'persistent' !
- Data persistence is one of the strength of DBs
- Two approaches
 - Take persistence to OO Languages
 - Object Database Systems – OQL
 - Take 'rich structures' to databases
 - Object-Relational Databases (ORDBMSs)

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

9

Knowledge Bases

- Knowledge Bases
 - Founded on mathematical logic based languages
 - Inferences / reasoning incorporated
 - *Every surgeon is a doctor; John is a surgeon and hence a doctor*
 - Data : modeled as collection of ‘statements’
 - Metadata: Terms/Vocabulary of the domain
 - Names of entity types of interest, properties of interest for these entities; binary relationships among entities
 - Modeled as Ontologies to capture domain knowledge
 - Can be viewed as graphs – ‘Knowledge Graphs’
 - More in the TAO course....

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

10

NoSQL Databases

- Non-relational databases / Not only SQL
 - Offer flexibility – no rigid row/col structure
 - Semi-structured + other simpler data models
 - Need: faster response to certain queries, quick data model updates, quick app development, horizontal scale-out ..
 - System ‘availability’ over ‘consistency’ of reads
 - (-ve) No standard query language to access data
 - Multiple types of DBs
 - JSON / XML, key-value, columnar, graph etc

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

11

Types of NoSQL DBs

- JSON / XML DBs
 - aka ‘Document-oriented’ / ‘document’ DBs
 - Semi-structured / non-homogenous data model
- Key-value Stores
 - Value – single or a collection of values (opaque)
 - Like hash-tables – give key, get value – simple
- Graph DBs – Nodes and labeled edges
 - Query language support – SPARQL etc
- Column-oriented DBs – fast access of col data

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

12

Course Organization

- Tutorials – Problem sessions – 5 or 6
 - Problem sheet – start of the class or a day before
 - Students to solve - discussion, questions – Ok
 - No marks – no judging; answers posted later
- Moodle based assessment tests – TBD
- Design and Dev assignment – group/individual
 - Running assignment – with 3 or 4 stages
- End-Sem exam – at the campus or KVs (?)
- Attendance – Google Meet tool

Prof P Sreenivasa Kumar, Department
of CS&E, IITM.

13

Database Systems

Introduction

Dr P Sreenivasa Kumar
 Professor
 CS&E Department
 IIT Madras

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

1

Introduction

What is a Database?

A collection of related pieces of data:

- Representing/capturing the information about a real-world enterprise or part of an enterprise.
- Collected and maintained to serve specific data management needs of the enterprise.
- Activities of the enterprise are supported by the database and continually update the database.

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

2

An Example

University Database:

Data about students, faculty, courses, research-laboratories, course registration/enrollment etc.
 Reflects the state of affairs of the academic aspects of the university.

Purpose: To keep an accurate track of the academic activities of the university.

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

3

Database Management System (DBMS)

A *general purpose* software system enabling:

- Creation of large disk-resident databases.
- Posing of data retrieval queries in a standard manner.
- Retrieval of query results efficiently.
- Concurrent use of the system by a large number of users in a consistent manner.
- Guaranteed availability of data irrespective of system failures.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

4

OS File System Storage Based Approach

- Files of records – used for data storage
 - data redundancy – wastage of space
 - maintaining consistency becomes difficult
- Record structures – hard coded into the programs
 - structure modifications – hard to perform
- Each different data access request (a query)
 - performed by a separate program
 - difficult to anticipate all such requests
- Creating the system
 - requires a lot of effort
- Managing concurrent access and failure recovery are difficult

Prof P Sreenivasa Kumar
Department of CS&E, IITM

5

DBMS Approach

DBMS

- separation of data and metadata
- flexibility of changing metadata
- program-data independence

Data access language

- standardized – SQL
- ad-hoc query formulation – easy

System development

- less effort required
- concentration on logical level design is enough
- components to organize data storage
 - process queries, manage concurrent access,
 - recovery from failures, manage access control
- are all available

Prof P Sreenivasa Kumar
Department of CS&E, IITM

6

Data Model

Collection of conceptual tools to describe the database at a certain level of abstraction.

- *Conceptual Data Model*
 - a high level description
 - useful for requirements understanding.
- *Representational Data Model*
 - describing the logical representation of data without giving details of physical representation.
- *Physical Data Model*
 - description giving details about record formats, file structures etc.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

7

E/R (Entity/Relationship) Model

- A conceptual level data model.
 - Provides the concepts of *entities*, *relationships* and *attributes*.
- The University Database Context*
Entities: *student*, *faculty member*, *course*, *departments* etc.
Relationships: *enrollment* relationship between student & course,
employment relationship between faculty member, department etc.
Attributes: *name*, *rollNumber*, *address* etc., of *student* entity,
name, *empNumber*, *phoneNumber* etc., of *faculty* entity etc.
More details will be given in the E/R Model Module.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

8

Representational Level Data Model

Relational Model : Provides the concept of a relation.

In the context of university database:

Relation name

Attributes

student						
SName	RollNumber	JoiningYear	BirthDate	Program	Dept	
Sriram	CS04B123	2004	15Aug1982	BTech	CS	
⋮	⋮	⋮	⋮	⋮	⋮	

Data tuple

Relation scheme: Attribute names of the relation.

Relation data/instance: set of data tuples.

More details will be given in Relational Data Model Module.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

9

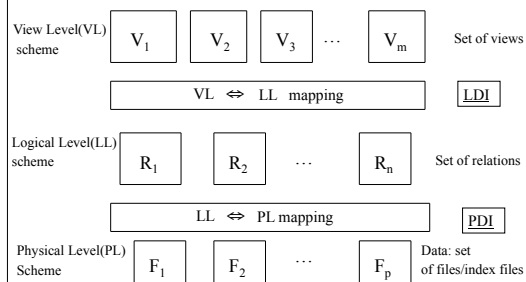
RDBMS: Data versus Schema or Meta-Data

- DBMS is generic in nature
 - not tied to a single database
 - capable of managing several databases at a time
- Data and schema are stored separately.
- In RDBMS context:
 - Schema – table names, attribute names with their data types for each table and constraints etc.
 - More details later...
- Database definition – setting up the skeleton structure
- Database Loading/populating – storing data

Prof P Sreenivasa Kumar
Department of CS&E, IITM

10

Abstraction Levels in a DBMS: Three-Schema Architecture



Prof P Sreenivasa Kumar
Department of CS&E, IITM

11

RDBMS: Three-schema Architecture(1/2)

View Level Schema

Each view describes an aspect of the database relevant to a particular group of users.

For instance, in the context of a library database:

- Books Purchase Section
- Issue/Returns Management Section
- Users Management Section

Each section views/uses a portion of the entire data.

Views can be set up for each section of users.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

12

RDBMS: Three-schema Architecture(2/2)

Logical Level Schema

- Describes the logical structure of the entire database.
- No physical level details are given.

Physical Level Schema

- Describes the physical structure of data in terms of record formats, file structures, indexes etc.

Remarks

- Views are optional
 - Can be set up if the DB system is very large and if easily identifiable user-groups exist
- The logical scheme is essential
- Modern RDBMS' s hide details of the physical layer

Prof P Sreenivasa Kumar
Department of CS&E, IITM

13

Physical Data Independence

The ability to modify physical level schema without affecting the logical or view level schema.

Performance tuning – modification at physical level
creating a new index etc.

Physical Data Independence – modification is localized

- achieved by suitably modifying PL-LL mapping.
- a very important feature of modern DBMS.

Three Schema Arch

Prof P Sreenivasa Kumar
Department of CS&E, IITM

14

Logical Data Independence

The ability to change the logical level scheme without affecting the view level schemes or application programs

Adding a new attribute to some relation

- no need to change the programs or views that don't require to use the new attribute

Deleting an attribute

- no need to change the programs or views that use the remaining data
- view definitions in VL-LL mapping only need to be changed for views that use the deleted attribute

Three-schema Architecture

Prof P Sreenivasa Kumar
Department of CS&E, IITM

15

Development Process of a Database System (1/2)

Step 1. Requirements collection

- *Data model requirements*
 - various pieces of data to be stored and the interrelationships.
 - presented using a *conceptual data model* such as E/R model.
- *Functional requirements*
 - various operations that need to be performed as part of running the enterprise.
 - acquiring a new book, enrolling a new user, issuing a book to the user, recording the return of a book etc.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

16

Development process of a database system (2/2)

Step 2. Convert the conceptual data model

into a representational level data model

- typically the relational data model.
- choose an RDBMS system and create the database.

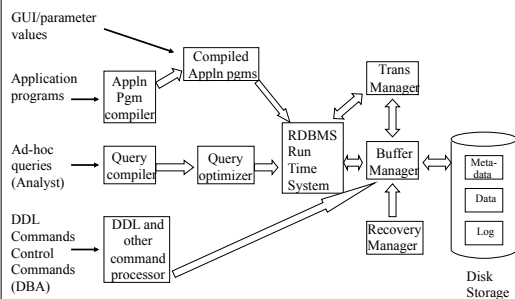
Step 3. Convert the functional requirements into application programs

- programs in a high-level language that use embedded SQL /API's to interact with the database and carry out the required tasks.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

17

Architecture of an RDBMS system



Prof P Sreenivasa Kumar
Department of CS&E, IITM

18

Architecture Details (1/3)

Disk Storage:

- Meta-data / schema
 - table definitions, view definitions, mappings
- Data – relation instances, index structures
 - statistics about data
- Log – record of database update operations
 - essential for failure recovery

DDL and other SQL command processor:

- (DDL – Data definition language part of SQL)
- Commands for relation scheme creation, constraints setting etc
- Commands for handling authorization and data access control

Prof P Sreenivasa Kumar
Department of CS&E, IITM

19

Architecture Details (2/3)

Query compiler

- Compiles
 - SQL adhoc queries
 - update / delete commands

Query optimizers

- Selects a near optimal plan for executing a query
- relation properties and index structures are utilized

Application Program Compiler

- Preprocess to separate embedded SQL commands
- Use host language compiler to compile rest of the program
- Integrate the compiled program with the libraries for SQL commands supplied by RDBMS

Prof P Sreenivasa Kumar
Department of CS&E, IITM

20

Architecture Details (3/3)

RDBMS Run Time System:

- Executes Compiled queries, Compiled application programs
- Interacts with Transaction Manager, Buffer Manager

Transaction Manager:

- Keeps track of start, end of each transaction
- Enforces concurrency control protocols

Buffer Manager:

- Manages disk space
- Implements paging mechanism

Recovery Manager:

- Takes control as restart after a failure
- Brings the system to a consistent state before it can be resumed

Prof P Sreenivasa Kumar
Department of CS&E, IITM

21

Roles for people in an Info System Management (1/2)

Naive users / Data entry operators

- Use the GUI provided by an application program
- Feed-in the data and invoke an operation
 - e.g., person at the train reservation counter, person at library issue / return counter
- No deep knowledge of the IS required

Application Programmers

- Embed SQL in a high-level language and develop programs to handle functional requirements of an IS
- Should thoroughly understand the logical schema or relevant views
- Meticulous testing of programs - necessary

Prof P Sreenivasa Kumar
Department of CS&E, IITM

22

Roles for people in an Info System management (2/2)

Sophisticated user / data analyst:

Uses SQL to generate answers for complex queries

DBA (Database Administrator)

Designing the logical scheme
Creating the structure of the entire database
Monitor usage and create necessary index structures to speed up query execution
Grant / Revoke data access permissions to other users etc.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

23

Text Books

- Ramez Elmasri and Shamkant B Navathe, *Fundamentals of Database Systems*, 6th Edition, Addison Wesley, 2011.
- Raghu Ramakrishnan and Johannes Gehrke, *Database Management Systems*, 3rd Edition, McGraw Hill, 2003.
- A Silberschatz, H F Korth and S Sudarshan, *Database System Concepts*, 6th Edition, 2013.
- H Garcia-Molina, J D Ullman, and Jennifer Widom, *Database Systems - The Complete Book*, Pearson Education, 2002.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

24

Entity-Relationship (E/R) Model

Dr P Sreenivasa Kumar
Professor
CS&E Dept IIT Madras

Entity-Relationship (E/R) Model

- Widely used conceptual level *data model*
 - proposed by Peter P Chen in 1970s
- Data model to describe the database system at the requirements collection stage
 - high level description.
 - easy to understand for the enterprise managers.
 - rigorous enough to be used for system building.
- Concepts available in the model
 - entities and attributes of entities.
 - relationships between entities.
 - diagrammatic notation.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

2

Entities

- *Entity* - a thing (animate or inanimate) of independent physical or conceptual existence and *distinguishable*.
In the University database context, an individual *student*, *faculty member*, a *class room*, a *course* are entities.
- *Entity Set* or *Entity Type*-
Collection of entities all having the same properties.
Student entity set – collection of all *student* entities.
Course entity set – collection of all *course* entities.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

3

Attributes

Each entity is described by a set of attributes/properties that have associated values

student entity

- *StudName* – name of the student.
- *RollNumber* – the roll number of the student.
- *Sex* – the gender of the student etc.

All entities in an Entity set/type have the same set of attributes.

Chosen set of attributes – amount of detail in modeling.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

4

Types of Attributes (1/2)

- Simple Attributes
 - having atomic or indivisible values.
 - example: *Dept* – a string
PhoneNumber – a ten digit number
- Composite Attributes
 - having several components in the value.
 - example: *Qualification* with components
(*DegreeName*, *Year*, *UniversityName*)
- Derived Attributes
 - Attribute value is dependent on some other attribute.
 - example: *Age* depends on *DateOfBirth*.
So age is a derived attribute.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

5

Types of Attributes (2/2)

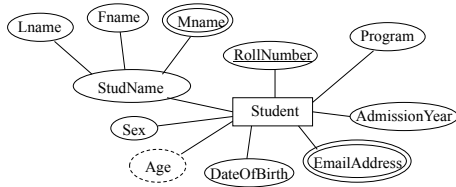
- Single-valued
 - having only one value rather than a set of values.
 - for instance, *PlaceOfBirth* – single string value.
- Multi-valued
 - having a set of values rather than a single value.
 - for instance, *CoursesEnrolled* attribute for student
EmailAddress attribute for student
PreviousDegree attribute for student.
- Attributes can be:
 - simple single-valued, simple multi-valued,
 - composite single-valued or composite multi-valued.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

6

Diagrammatic Notation for Entities

entity - rectangle
attribute - ellipse connected to rectangle
multi-valued attribute - double ellipse
composite attribute - ellipse connected to ellipse
derived attribute - dashed ellipse



Prof P Sreenivasa Kumar
Department of CS&E, IITM

7

Domains of Attributes

Each attribute takes values from a set called its *domain*

For instance, *studentAge* – {17,18, ..., 55}

HomeAddress – character strings of length 35

Domain of composite attributes –

cross product of domains of component attributes

Domain of multi-valued attributes –

set of subsets of values from the basic domain

Prof P Sreenivasa Kumar
Department of CS&E, IITM

8

Entity Sets and Key Attributes

- *Key* – an attribute or a collection of attributes whose value(s) uniquely identify an entity in the entity set.
- For instance,
 - *RollNumber* - Key for *Student* entity set
 - *EmpID* - Key for *Faculty* entity set
 - *HostelName, RoomNo* - Key for *Student* entity set (assuming that each student gets to stay in a single room)
- A key for an entity set may have more than one attribute.
- An entity set may have more than one key.
- Keys can be determined only from the meaning of the attributes in the entity type.
 - Determined by the designers

Prof P Sreenivasa Kumar
Department of CS&E, IITM

9

Relationships

- When two or more entities are associated with each other, we have an instance of a *Relationship*.
- E.g.: *student* Ramesh *enrolls* in Discrete Mathematics *course*
- Relationship *enrolls* has *Student* and *Course* as the *participating* entity sets.
- Formally, $enrolls \subseteq Student \times Course$
 - $(s,c) \in enrolls \Leftrightarrow$ Student 's' has enrolled in Course 'c'
 - Tuples in *enrolls* – relationship instances
 - *enrolls* is called a relationship Type/Set.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

10

Degree of a relationship

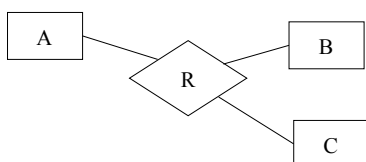
- *Degree* : the number of participating entities.
 - Degree 2: *binary*
 - Degree 3: *ternary*
 - Degree n: *n-ary*
- Binary relationships are very common and widely used.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

11

Diagrammatic Notation for Relationships

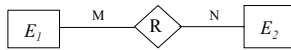
- Relationship – diamond shaped box
 - Rectangle of each participating entity is connected by a line to this diamond. Name of the relationship is written in the box.



Prof P Sreenivasa Kumar
Department of CS&E, IITM

12

Binary Relationships and Cardinality Ratio



- The *maximum* number of entities from E_2 that an entity from E_1 can possibly be associated thru R (and vice-versa) determines the *cardinality ratio* of R .
- Four possibilities are usually specified:
 - *one-to-one* (1:1)
 - *one-to-many* (1:N)
 - *many-to-one* (N:1)
 - *many-to-many* (M:N)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

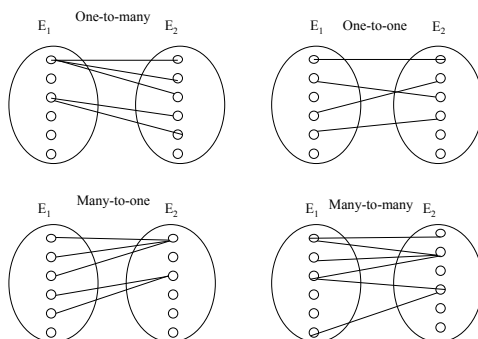
13

Cardinality Ratios

- *One-to-one*: An E_1 entity may be associated with at most one E_2 entity and similarly an E_2 entity may be associated with at most one E_1 entity.
- *One-to-many*: An E_1 entity may be associated with many E_2 entities whereas an E_2 entity may be associated with at most one E_1 entity.
- *Many-to-one*: An E_2 entity may be associated with many E_1 entities whereas an E_1 entity may be associated with at most one E_2 entity.
- *Many-to-many*: Many E_1 entities may be associated with a single E_2 entity and a single E_1 entity may be associated with many E_2 entities.

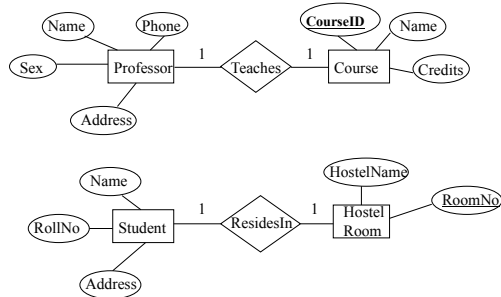
Prof P Sreenivasa Kumar
Department of CS&E, IITM

14



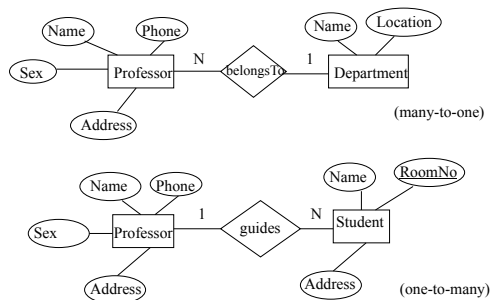
Prof P Sreenivasa Kumar
Department of CS&E, IITM

15

Cardinality Ratio – example (*one-to-one*)

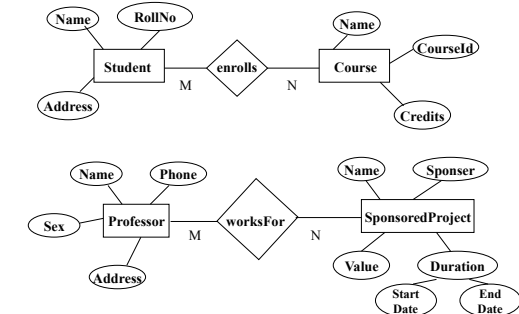
Prof P Sreenivasa Kumar
Department of CS&E, IITM

16

Cardinality Ratio – example (*many-to-one/one-to-many*)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

17

Cardinality Ratio – example (*many-to-many*)

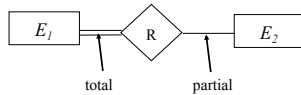
Prof P Sreenivasa Kumar
Department of CS&E, IITM

18

Participation Constraints

- An entity set may participate in a relation either *totally* or *partially*.
 - Total participation:** Every entity in the set is involved in some association (or tuple) of the relationship.
 - Partial participation:** Not all entities in the set are involved in association (or tuples) of the relationship.

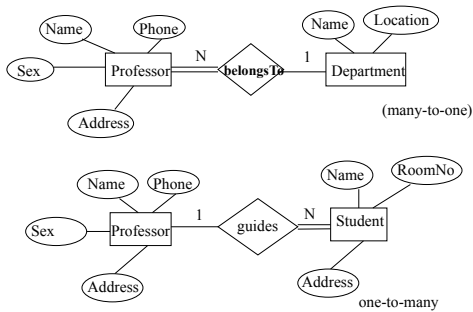
Notation:



Prof P Sreenivasa Kumar
Department of CS&E, IITM

19

Example of total/partial Participation



Prof P Sreenivasa Kumar
Department of CS&E, IITM

20

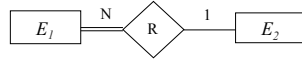
Structural Constraints

- Cardinality Ratio and Participation Constraints are together called *Structural Constraints*.
- They are called *constraints* as the *data* must satisfy them to be consistent with the requirements.
- Min-Max notation:** pair of numbers (m, n) placed on the line connecting an entity to the relationship.
- m :** the minimum number of times a particular entity *must appear* in the relationship tuples at any point of time
 - 0 – partial participation
 - ≥ 1 – total participation
- n :** similarly, the maximum number of times a particular entity *can appear* in the relationship tuples at any point of time

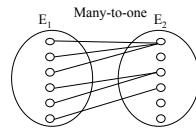
Prof P Sreenivasa Kumar
Department of CS&E, IITM

21

Comparing the Notations



is equivalent to



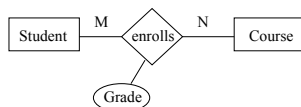
Prof P Sreenivasa Kumar
Department of CS&E, IITM

22

Attributes for Relationship Types

Relationship types can also have attributes.

- properties of the association of entities.

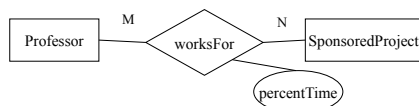
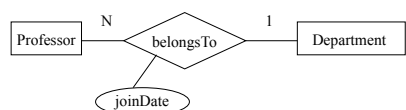


- grade* gives the letter grade (S,A,B, etc.) earned by the student for a course.
- neither an attribute of *student* nor that of *course*.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

23

Attributes for Relationship Types – More Examples

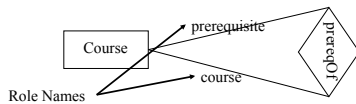


Prof P Sreenivasa Kumar
Department of CS&E, IITM

24

Recursive Relationships and Role Names

- Recursive relationship: An entity set relating to itself gives rise to a *recursive* relationship
- E.g., the relationship *prereqOf* is an example of a recursive relationship on the entity *Course*
- Role Names – used to specify the exact role in which the entity participates in the relationships
 - Essential in case of recursive relationships
 - Can be optionally specified in non-recursive cases



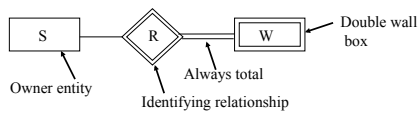
Prof P Sreenivasa Kumar
Department of CS&E, IITM

25

Weak Entity Sets

Weak Entity Set: An entity set whose members owe their existence to some entity in a *strong entity set*.

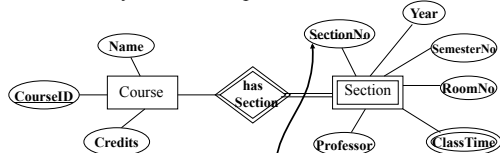
- weak entities are not of independent existence.
- each weak entity is associated with some entity of the *owner* entity set through a special relationship
- weak entity set may not have a key attribute
- the *owner* entity might itself be a weak entity
- identifying relationship may not always be binary



Prof P Sreenivasa Kumar
Department of CS&E, IITM

26

Weak Entity Sets - Example

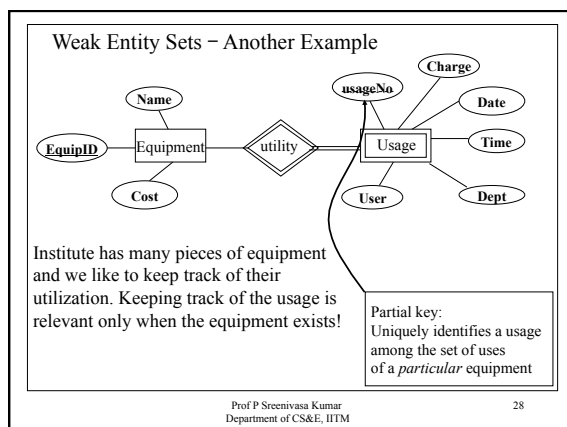


A popular course may have several sections each taught by a different professor and having its own class room and meeting times

Partial key:
Uniquely identifies a section among the set of sections of a particular course

Prof P Sreenivasa Kumar
Department of CS&E, IITM

27



Complete Example for E/R schema: Specifications (1/2)

In an educational institute, there are several departments and each student belongs to one of them. Each department has a unique department number, a name, a location, phone number and is headed by a professor. Professors have a unique employee Id, name and a phone number. A professor works for exactly one department.

We like to keep track of the following details regarding students: name, unique roll number, sex, phone number, date of birth, age and one or more email addresses. Students have a local address consisting of the hostel name and the room number. They also have home address consisting of house number, street, city and PIN. It is assumed that all students reside in the hostels.

29

Prof P Sreenivasa Kumar
Department of CS&E, IITM

Complete Example for E/R schema: Specifications (2/2)

A course taught in a semester of the year is called a *section*. There can be several sections of the same course in a semester; these are identified by the *section number*. Each section is taught by a professor and has its own timings and a room to meet. Students enroll for several sections in a semester.

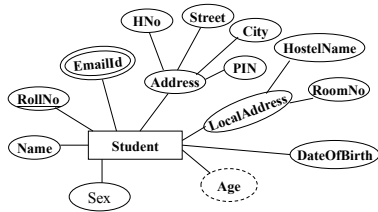
Each course has a name, number of credits and the department that offers it. A course may have other courses as pre-requisites i.e., courses to be completed before it can be enrolled in.

Professors also undertake research projects. These are sponsored by funding agencies and have a specific start date, end date and amount of money given. More than one professor can be involved in a project. Also a professor may be simultaneously working on several projects. A project has a unique *projectId*.

30

Prof P Sreenivasa Kumar
Department of CS&E, IITM

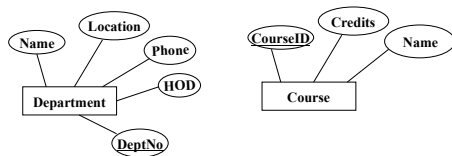
Entities - Student



Prof P Sreenivasa Kumar
Department of CS&E, IITM

31

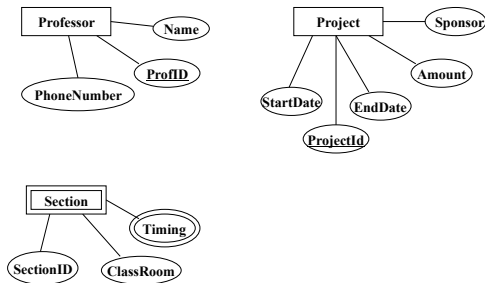
Entities – Department and Course



Prof P Sreenivasa Kumar
Department of CS&E, IITM

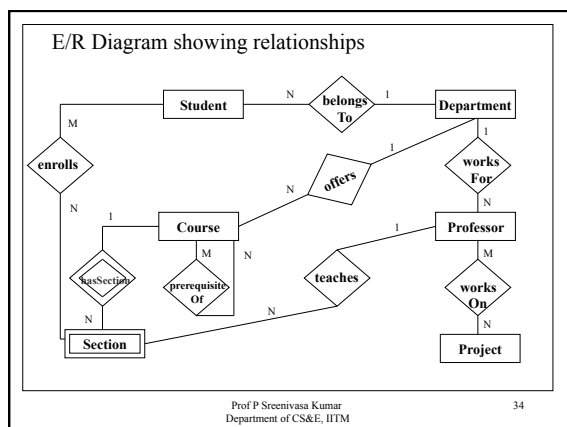
32

Entities – Professor, Project and Sections



Prof P Sreenivasa Kumar
Department of CS&E, IITM

33



Design Choices: Attribute versus Relationship

- Should *offering department* be an attribute of a course or should we create a relationship between Course and Dept entities called, say, *offers* ?
 - Later approach is preferable when the necessary entity, in this case the Department, already exists.
- Should *class room* be an attribute of Section or should we create an entity called ClassRoom and have a relationship, say, *meetsIn*, connecting Section and ClassRoom?
 - In this case, the option of making classRoom as an attribute of Section is better as we do not want to give a lot of importance to class room and make it a an *entity*.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

35

Design Choices:

Weak entity versus composite multi-valued attributes

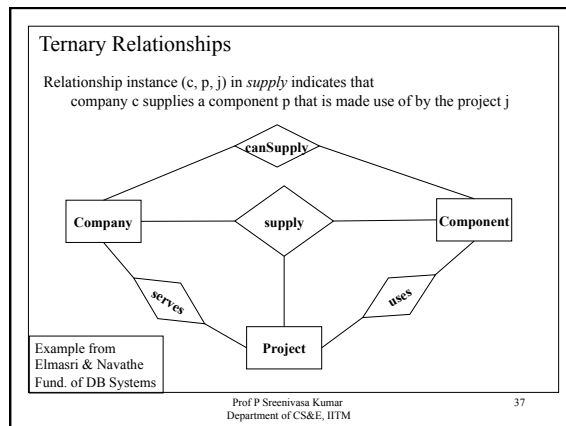
- Note that a *section* could also be modeled as a composite multi-valued *attribute* of Course entity.
 - However, if so, *section* can not participate in relationships, such as, *enrolls* with Student entity.
- In general, if a thing, even though not of independent existence, participates in other relationships on its own, it is best captured as a *weak entity*.
 - If the above is not the case, composite multi-valued attribute may be enough.

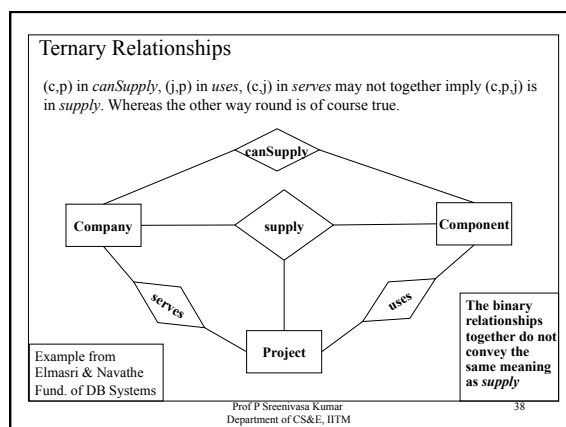
The enterprise/organization

- whose domain model is being built will not usually figure in the model!

Prof P Sreenivasa Kumar
Department of CS&E, IITM

36





Tuple Relational Calculus (TRC)

Introduction

Procedural Query language

- query specification involves giving a step by step process of obtaining the query result
e.g., relational algebra
- usage calls for detailed knowledge of the operators involved
- difficult for the use of non-experts

Declarative Query language

- query specification involves giving the logical conditions the results are required to satisfy
- easy for the use of non-experts

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

1

Predicates

We have studied “Predicate Logic” in Discrete Maths course

What is a “predicate”?

A predicate is an expression with place-holders (aka variables)

- it takes a Boolean value when values are substituted for variables

lessThan(x,y): lessThan(2,5) is true; lessThan(5,1) is false

livesIn(p,c): livesIn(SindhuPV, Hyderabad) - true

livesIn(AvaniLekhara, Jaipur) - true

livesIn(AdarPoonawala, Chennai) - false

One can have n -place predicates also...

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

2

Relations and Predicates

Can we interpret

“relation names” of a relational model as predicates??

Recall: DB Relation – finite set of tuples

Tuple – sequence of values;

each value corresponds to an attribute
and comes from a domain of atomic values

Each DB relation name with n attributes

- can be thought of as an n -place predicate

We make Closed-World Assumption (CWA) where...

- tuples in the relation instance make the predicate true
- those not present in the instance make it false

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

3

TRC – a declarative query language

Tuple variable – associated with a relation
(called the *range relation*)

- takes tuples as its values
- t : tuple variable over relation r with scheme $R(A,B,C)$
 $t.A$ stands for value of column/attribute A in t etc

TRC Query – basic form:

$\{ t_1.A_{i_1}, t_2.A_{i_2}, \dots, t_m.A_{i_m} \mid \theta \}$

predicate calculus/logic expression
involving tuple variables

$t_1, t_2, \dots, t_m, t_{m+1}, \dots, t_s$

- specifies the condition to be satisfied

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

4

An example TRC query

student (rollNo, name, degree, year, sex, deptNo, advisor)
department (deptId, name, hod, phone)

Obtain the rollNo, name of all girl students
in the Maths Dept (deptId = 2)

$\{ s.rollNo, s.name \mid student(s) \wedge s.sex = 'F' \wedge s.deptNo = 2 \}$

attributes
required in
the result

This predicate is true whenever
value of s is a tuple from the
Student instance, false otherwise

In general, if t is a tuple variable with range
relation r , $r(t)$ is taken as a predicate which
is true if and only if the value of t is a tuple in r

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

5

General form of the condition in TRC queries

Atomic expressions are the following:

1. $r(t)$ -- true if t is a tuple in the relation instance r
2. $t_1.A_i <compOp> t_2.A_j$ $compOp$ is one of $\{<, \leq, >, \geq, =, \neq\}$
3. $t.A_i <compOp> c$ c is a constant of appropriate type

Composite expressions:

1. Any atomic expression
2. $F_1 \wedge F_2, F_1 \vee F_2, \neg F_1$ where F_1 and F_2 are expressions
3. $(\forall t)(F), (\exists t)(F)$ where F is an expression
and t is a tuple variable

Free Tuple Variable – a variable that is not quantified

Bound Tuple Variable – a quantified variable

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

6

Interpretation of the query in TRC

All possible tuple assignments to the free variables in the query are considered.

For any specific assignment,
if the expression to the right of the vertical bar evaluates to true,
that combination of tuple values
would be used to produce a tuple in the result relation.

While producing the result tuple, the values of the attributes for the
corresponding tuple variables as specified on the left side of the
vertical bar would be used.

Note: The only free variables are the ones that appear to the left
of the vertical bar

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

7

Example TRC queries

Obtain the rollNo, name of all girl students in the
Maths Dept

```
{s.rollNo, s.name | student(s) ^ s.sex='F' ^  
  (∃ d)(department(d) ^ d.name='Maths'  
    ^ d.deptId = s.deptNo)}
```

s : free tuple variable d : existentially bound tuple variable

Existentially or universally quantified tuple variables can be used
on the RHS of the vertical bar to specify query conditions

Attributes of free (or unbound) tuple variables can be used on LHS
of vertical bar to specify attributes required in the results

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

8

Example Relational Scheme

student (rollNo, name, degree, year, sex, deptNo, advisor)

department (deptId, name, hod, phone)

professor (empId, name, sex, startYear, deptNo, phone)

course (courseId, cname, credits, deptNo)

enrollment (rollNo, courseId, sem, year, grade)

teaching (empId, courseId, sem, year, classRoom)

prerequisite (preReqCourse, courseId)

Q2

Q3

Q4

Q5

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

9

Example queries in TRC (1/5)

1) Determine the departments that do not have any girl students

student (rollNo, name, degree, year, sex, deptNo, advisor)
department (deptId, name, hod, phone)

```
{d.name | department(d) ^  
  ¬(∃ s)(student(s) ^  
    s.sex = 'F' ^ s.deptNo = d.deptId)}
```

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

10

Examples queries in TRC (2/5)

Schema

2) Obtain the names of courses enrolled by student named Mahesh

```
{c.name | course(c) ^  
  (∃ s)(∃ e)(student(s) ^ enrollment(e)  
    ^ s.name = "Mahesh"  
    ^ s.rollNo = e.rollNo  
    ^ c.courseId = e.courseId)}
```

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

11

Examples queries in TRC (3/5)

Schema

3) Get the names of students who have scored 'S' in all subjects they have enrolled. Assume that every student is enrolled in at least one course.

```
{s.name | student(s) ^  
  (∀ e)((enrollment(e) ^  
    e.rollNo = s.rollNo) → e.grade = 'S')}
```

Person P with all S grades:

For enrollment tuples not having her roll number, LHS (of \rightarrow) is false.

For enrollment tuples having her roll number, LHS is true, RHS is also true.

So the implication is *true* for all e tuples.

Person Q with some non-S grades:

For enrollment tuples not having her roll number, LHS is false.

For enrollment tuples having her roll number, LHS is true, but RHS is false for at least one tuple.

So the implication is *not true* for at least one tuple.

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

12

Examples queries in TRC (4/5)

Schema

- 4) Get the names of students who have taken at least one course taught by their advisor.

```
{s.name | student(s) ^
  (∃e)(∃t)(enrollment(e) ^ teaching(t) ^
    e.courseId = t.courseId ^
    e.sem = t.sem ^ e.year = t.year ^
    e.rollNo = s.rollNo ^ t.empId = s.advisor)}
```

- 5) Display the departments whose HODs taught at least one course in the odd semester of 2019.

```
{d.name | department(d) ^ (∃t)(teaching(t) ^
  t.empId = d.hod ^
  t.sem = 'odd' ^ t.year = '2019')}
```

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

13

Examples queries in TRC (5/5)

Schema

- 6) Determine the students who are enrolled for **every** course taught by Prof Ramanujam. Assume that Prof Ramanujam teaches at least one course.

Free tuple variable?
Student

every course is such that if it is a course taught by Ramanujam, the student required in the result has enrolled for it..

Bound tuple variables?
course - universal quantifier

professor, teaching..
enrollment
- existential quantifier

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

14

Examples queries in TRC (5/5)

Schema

- 6) Determine the students who are enrolled for **every** course taught by Prof Ramanujam. Assume that Prof Ramanujam teaches at least one course.

```
1. {s.rollNo | student (s) ^
2.   (∀c)(course (c) ^
3.     ((∃t),(∃p)( teaching(t) ^ professor(p) ^
4.       t.courseId = c.courseId ^
5.       p.name = "Ramanujam" ^
6.       p.empId = t.empId ) →
7.       (∃e) (enrollment(e) ^
8.         e.courseId = c.courseId ^
9.         e.rollNo = s.rollNo ^
10.        e.sem = t.sem ^
11.        e.year = t.year ))}
12.
```

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

15

Problem with unrestricted use of Negation

What is the result of the query:

$\{s.rollNo \mid \neg student(s)\} ?$

Infinite answers !!

Unsafe TRC expression :

Any expression whose result uses “constants / values” that do not appear in the instances of any of the database relations.

Unsafe expressions are to be avoided while specifying TRC queries.

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

16

Expressive power of TRC and Relational Algebra

It can be shown that

both Tuple Relational Calculus and Relational Algebra have the same expressive power

A query can be formulated in (safe) TRC if and only if it can be formulated in RA

Both *can not* be used to formulate queries involving *transitive closure*

- find all direct or indirect pre-requisites of a course
- find all subordinates of a specific employee etc.

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

17

Try this query - 1

List the name, roll number of students along with the courses (course number only) in which they got an S grade.

student (rollNo, name, degree, year, sex, deptNo, advisor)
enrollment (rollNo, courseId, sem, year, grade)

$\{s.rollNo, s.name, e.courseId \mid$
 $student(s) \wedge enrollment(e)$
 $s.rollNo = e.rollNo \wedge$
 $e.grade = "S" \}$

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

18

Try this query - 2

List the name, roll number of male students being advised by lady professors along with the name of the advisor.

student (rollNo, name, degree, year, sex, deptNo, advisor)
professor (empId, name, sex, startYear, deptNo, phone)

```
{s.rollNo, s.name, p.name |  
  student(s) ^ professor(p)  
  s.sex = "M" ^ p.sex = "F" ^  
  s.advisor = p.empId }
```

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

19

Try this query - 3

List the name, course number of all the prerequisites of the course "Data Mining".

course (courseId, cname, credits, deptNo)
preRequisite (preReqCourse, courseId)

```
{c.courseId, c.cname |  
  course(c) ^  
  (∃d)(∃q)( course(d) ^ d.cname = "Data Mining" ^  
    preRequisite(q) ^  
    q.courseId = d.courseId ^  
    q.preReqCourse = c.courseId )  
}
```

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

20

Relational Data Model

Introduction

- Proposed by Edgar F Codd (1923-2003) in the early seventies [Turing Award – 1981]
- Most of the modern DBMS use the *relational* data model.
- Simple and elegant model with a mathematical basis.
- Led to the development of a theory of data dependencies and database design.
- Relational algebra operations – crucial role in query optimization and execution.
- Laid the foundation for the development of
 - Tuple relational calculus and then
 - Database standard SQL

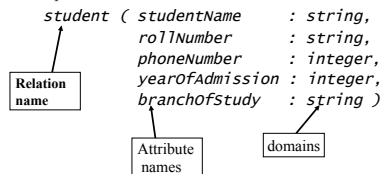
Prof P Sreenivasa Kumar
Department of CS&E, IITM

1

Relation Scheme

- Consists of relation name, and a set of attributes or field names or column names. Each attribute has an associated domain.

- Example:*



- Domain* – set of *atomic* (or *indivisible*) values – data type

Prof P Sreenivasa Kumar
Department of CS&E, IITM

2

Relation Instance

- A finite *set of tuples* constitute a relation instance.
- A tuple of the relation with scheme $R = (A_1, A_2, \dots, A_m)$ is an ordered sequence of values (v_1, v_2, \dots, v_m) such that $v_i \in \text{domain}(A_i), 1 \leq i \leq m$

student

studentName	rollNumber	yearOf Admission	phoneNumber	branch Of Study
Ravi Teja	CS05B015	2005	9840110489	CS
Rajesh	CS04B125	2004	9840110490	CS

No duplicate tuples (or rows) in a relation instance.

We shall later see that in SQL, duplicate rows would be allowed in tables.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

3

Another Relation Example

enrollment (studentName, rollNo, courseNo, sectionNo)

enrollment

studentName	rollNumber	courseNo	sectionNo
Rajesh	CS04B125	CS3200	2
Rajesh	CS04B125	CS3700	1
Suresh	CS04B130	CS3200	2

Prof P Sreenivasa Kumar
Department of CS&E, IITM

4

Keys for a Relation (1/2)

- **Key:** A set of attributes K, whose values uniquely identify a tuple in any instance. And none of the proper subsets of K has this property
Example: {rollNumber} is a key for *student* relation.
{rollNumber, name} – values can uniquely identify a tuple
 - but the set is not *minimal*
 - not a Key
- A key can not be determined from any particular instance data
 - it is an intrinsic property of a scheme
 - it can only be determined from the meaning of attributes

Prof P Sreenivasa Kumar
Department of CS&E, IITM

5

Keys for a Relation (2/2)

- A relation can have more than one key.
- Each of the keys is called a *candidate* key
Example: *book* (isbnNo, authorName, title, publisher, year)
(Assumption : books have only one author)
Keys: {isbnNo}, {authorName, title}
- A relation has at least one key
- the set of all attributes, in case no proper subset is a key.
- **Superkey:** A set of attributes that contains a key as a subset.
 - A key can also be defined as a *minimal superkey*
- **Primary Key:** One of the candidate keys chosen for indexing purposes (More details later...)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

6

Relational Database Scheme and Instance

Relational database scheme: D consist of a finite no. of relation schemes and a set I of integrity constraints.

Integrity constraints: Necessary conditions to be satisfied by the data values in the relational instances so that the set of data values constitute a meaningful database

- domain constraints
- key constraints
- referential integrity constraints

Database instance: Collection of relational instances satisfying the integrity constraints.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

7

Domain and Key Constraints

- **Domain Constraints:** Attributes have associated domains

Domain – set of atomic data values of a specific type.

Constraint – stipulates that the actual values of an attribute in any tuple must belong to the declared domain.

- **Key Constraint:** Relation scheme – associated keys

Constraint – if K is supposed to be a key for scheme R , any relation instance r on R should not have two tuples that have identical values for attributes in K .

Also, none of the key attributes can have null value.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

8

Foreign Keys

- Tuples in one relation, say $r_1(R_1)$, often need to refer to tuples in another relation, say $r_2(R_2)$

- to capture relationships between entities

- Primary Key of $R_2 : K = \{B_1, B_2, \dots, B_j\}$

- A set of attributes $F = \{A_1, A_2, \dots, A_i\}$ of R_1 such that

$\text{dom}(A_i) = \text{dom}(B_j), \quad 1 \leq i \leq j$ and
whose values are used to refer to tuples in r_2
is called a *foreign key* in R_1 referring to R_2 .

- R_1, R_2 can be the same scheme also.
- There can be more than one foreign key in a relation scheme

Prof P Sreenivasa Kumar
Department of CS&E, IITM

9

Foreign Key – Examples (1/2)

Foreign key attribute *deptNo* of *course* relation refers to
Primary key attribute *deptID* of *department* relation

Course

courseId	name	credits	deptNo
CS635	ALGORITHMS	3	1
CS636	A.I	4	1
ES456	D.S.P	3	2
ME650	AERO DYNAMICS	3	3

Department

deptId	name	hod	phone
1	COMPUTER SCIENCE	CS01	22576235
2	ELECTRICAL ENGG	ES01	22576234
3	MECHANICAL ENGG	ME01	22576233

courseId	name	credits	deptNo
deptId	name	hod	phone

Prof P Sreenivasa Kumar
Department of CS&E, IITM

10

Foreign Key – Examples(2/2)

It is possible for a foreign key in a relation
to refer to the primary key of the relation itself

An Example:

univEmployee (empNo, name, sex, salary, dept, reportsTo)

reportsTo is a foreign key referring to *empNo* of the same relation

Every employee in the university reports to some other
employee for administrative purposes
- except the *vice-chancellor*, of course!

Prof P Sreenivasa Kumar
Department of CS&E, IITM

11

Referential Integrity Constraint (RIC)

- Let F be a foreign key in scheme R_1 referring to scheme R_2 and let K be the primary key of R_2 .
- RIC**: any relational instances r_1 on R_1 and r_2 on R_2 must be s.t for any tuple t in r_1 , either its F -attribute values are all *null* or they are identical to the K -attribute values of *some* tuple in r_2 .
- RIC ensures that references to tuples in r_2 are for *currently existing* tuples.
 - That is, there are no *dangling* references.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

12

Referential Integrity Constraint (RIC) - Example

COURSE				DEPARTMENT			
courseId	name	credits	deptNo	deptId	name	hod	phone
CS635	ALGORITHMS	3	1	1	COMPUTER SCIENCE	CS01	22576235
CS636	A.I	4	1	2	ELECTRICAL ENGG.	ES01	22576234
ES456	D.S.P	3	2	3	MECHANICAL ENGG.	ME01	22576233
ME650	AERO DYNAMICS	3	3				
CE751	MASS TRANSFER	3	4				

The course CE751 refers to a non-existent department 4 and thus violates the RIC

Prof P Sreenivasa Kumar
Department of CS&E, IITM

13

Example Relational Scheme

student (rollNo, name, degree, year, sex, deptNo, advisor)
degree is the program (B Tech, M Tech, M S, Ph D etc)
 for which the student has joined.
year is the year of admission and
advisor is the Empld of a faculty member identified as
 the student's advisor.

department (deptId, name, hod, phone)
phone is that of the department's office.

professor (empld, name, sex, startYear, deptNo, phone)
startYear is the year when the faculty member has
 joined the department *deptNo*.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

14

Example Relational Scheme

course (courseId, cname, credits, deptNo)
deptNo indicates the department that offers the course.

enrollment (rollNo, courseId, sem, year, grade)
sem can be either "odd" or "even" indicating the two
 semesters of an academic year.
 The value of *grade* will be null for the current semester
 and non-null for past semesters.

teaching (empld, courseId, sem, year, classRoom)

preRequisite (preReqCourse, courseId)
 Here, if (c1, c2) is a tuple, it indicates that c1 should be
 successfully completed before enrolling for c2.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

15

Example Relational Scheme

student (rollNo, name, degree, year, sex, deptNo, advisor)

department (deptId, name, hod, phone)

professor (empId, name, sex, startYear, deptNo, phone)

course (courseId, cname, credits, deptNo)

enrollment (rollNo, courseId, sem, year, grade)

teaching (empId, courseId, sem, year, classRoom)

preRequisite (preReqCourse, courseId)

queries-1
queries-2
queries-3
XProd
TCQuery

Prof P Sreenivasa Kumar
Department of CS&E, IITM

16

Example Relational Scheme with RICs shown

student (rollNo, name, degree, year, sex, deptNo, advisor)

department (deptId, name, hod, phone)

professor (empId, name, sex, startYear, deptNo, phone)

course (courseId, cname, credits, deptNo)

enrollment (rollNo, courseId, sem, year, grade)

teaching (empId, courseId, sem, year, classRoom)

preRequisite (preReqCourse, courseId)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

17

Relational Algebra

- A set of operators (unary and binary) that take relation instances as arguments and return new relations.
- Gives a procedural method of specifying a retrieval query.
- Forms the core component of a relational query engine.
- SQL queries are internally translated into RA expressions.
- Provides a framework for query optimization.

RA operations: *select* (σ), *project* (π), *cross product* (\times),
union (\cup), *intersection* (\cap), *difference* ($-$), *join* (\bowtie)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

18

The *select* Operator

- Unary operator.
- can be used to *select* those tuples of a relation that satisfy a given condition.
- *Notation:* $\sigma_{\theta}(r)$
 σ : select operator (read as *sigma*)
 θ : selection condition
 r : relation name
- Result: a relation with the same schema as r consisting of the tuples in r that satisfy condition θ
- Select operation is commutative:
 $\sigma_{c_1}(\sigma_{c_2}(r)) = \sigma_{c_2}(\sigma_{c_1}(r))$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

19

Selection Condition

- *Select condition:*
Basic condition or Composite condition
- *Basic condition:*
Either $A_i <\text{compOp}> A_j$ or $A_i <\text{compOp}> c$
- *Composite condition:*
Basic conditions combined with logical operators AND, OR and NOT appropriately.
- *Notation:*
 $<\text{compOp}>$: one of $<, \leq, >, \geq, =, \neq$
 A_i, A_j : attributes in the scheme R of r
 c : constant of appropriate data type

Prof P Sreenivasa Kumar
Department of CS&E, IITM

20

Examples of *select* expressions

1. Obtain information about a professor with name "Giridhar"

$\sigma_{\text{name} = \text{"Giridhar"}}(\text{professor})$

2. Obtain information about professors who joined the university between 1980 and 1985, both inclusive

$\sigma_{\text{startYear} \geq 1980 \wedge \text{startYear} \leq 1985}(\text{professor})$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

21

The *project* Operator

- Unary operator.
- Can be used to keep only the required attributes of a relation instance and throw away others.
- *Notation:* $\pi_{A_1, A_2, \dots, A_k}(r)$ where A_1, A_2, \dots, A_k is a list L of desired attributes in the scheme of r .
- **Result** = $\{ (v_1, v_2, \dots, v_k) \mid v_i \in \text{dom}(A_i), 1 \leq i \leq k \text{ and there is some tuple } t \text{ in } r \text{ s.t. } t.A_1 = v_1, t.A_2 = v_2, \dots, t.A_k = v_k \}$
- If $r_1 = \pi_L(r_2)$ then scheme of r_1 is L

Prof P Sreenivasa Kumar
Department of CS&E, IITM

22

Examples of *project* expressions

student

rollNo	name	degree	year	sex	deptNo	advisor
CS04S001	Mahesh	M.S	2004	M	1	CS01
CS03S001	Rajesh	M.S	2003	M	1	CS02
CS04M002	Piyush	M.E	2004	M	1	CS01
ES04M001	Deepak	M.E	2004	M	2	ES01
ME04M001	Lalitha	M.E	2004	F	3	ME01
ME03M002	Mahesh	M.S	2003	M	3	ME01

$\pi_{\text{rollNo, name}}(\text{student})$

rollNo	name
CS04S001	Mahesh
CS03S001	Rajesh
CS04M002	Piyush
ES04M001	Deepak
ME04M001	Lalitha
ME03M002	Mahesh

$\pi_{\text{name}}(\sigma_{\text{degree} = \text{"M.S"}}(\text{student}))$

name
Mahesh
Rajesh

Note: Mahesh is displayed only once because project operation results in a set.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

23

Size of *project* expression result

- If $r_1 = \pi_L(r_2)$ then scheme of r_1 is L
- What about the number of tuples in r_1 ?
- Two cases arise:
 - Projection List L contains some key of r_2
 - Then $|r_1| = |r_2|$
 - Projection List L does not contain any key of r_2
 - Then $|r_1| \leq |r_2|$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

24

Set Operators on Relations

- As relations are sets of tuples, set operations are applicable to them; but not in all cases.
- Union Compatibility** : Consider two schemes R_1, R_2 where $R_1 = (A_1, A_2, \dots, A_k); R_2 = (B_1, B_2, \dots, B_m)$
- R_1 and R_2 are called *union-compatible* if
 - $k = m$ and
 - $\text{dom}(A_i) = \text{dom}(B_i)$ for $1 \leq i \leq k$
- Set operations** – *union, intersection, difference*
 - Applicable to two relations if their schemes are union-compatible
- If $r_3 = r_1 \cup r_2$, scheme of r_3 is R_1 (as a convention)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

25

Set Operations

r_1 - relation with scheme R_1
 r_2 - relation with scheme R_2 - union compatible with R_1

$$r_1 \cup r_2 = \{t \mid t \in r_1 \text{ or } t \in r_2\}$$

$$r_1 \cap r_2 = \{t \mid t \in r_1 \text{ and } t \in r_2\}$$

$$r_1 - r_2 = \{t \mid t \in r_1 \text{ and } t \notin r_2\}$$

By convention, in all the cases, the scheme of the result is that of the first operand i.e r_1 .

Prof P Sreenivasa Kumar
Department of CS&E, IITM

26

Cross product Operation

r_1	A_1	A_2	...	A_m	$r_1 \times r_2$			
	A_1	A_2	...	A_m	B_1	B_2	...	B_n
a_{11}	a_{12}	...	a_{1m}	a_{11}	a_{12}	...	a_{1m}	b_{11}
a_{21}	a_{22}	...	a_{2m}	a_{21}	a_{22}	...	a_{2m}	b_{21}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
a_{s1}	a_{s2}	...	a_{sm}	a_{s1}	a_{s2}	...	a_{sm}	b_{s1}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
b_{11}	b_{12}	...	b_{1n}	b_{11}	b_{12}	...	b_{1n}	b_{11}
b_{21}	b_{22}	...	b_{2n}	b_{21}	b_{22}	...	b_{2n}	b_{21}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
b_{t1}	b_{t2}	...	b_{tn}	b_{t1}	b_{t2}	...	b_{tn}	b_{t1}
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
b_{m1}	b_{m2}	...	b_{mn}	b_{m1}	b_{m2}	...	b_{mn}	b_{m1}

$r_1 : s \text{ tuples}$ $r_2 : t \text{ tuples}$ $r_1 \times r_2 : s \times t \text{ tuples}$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

27

Example Query using *cross product*

Obtain the list of professors (Id and Name) along with the *name* of their respective departments

- Info is present in two relations: professor and department

Schema

- $\text{profDetail}(\text{eId}, \text{pname}, \text{deptno}) \leftarrow \pi_{\text{empId}, \text{name}, \text{deptNo}}(\text{professor})$
- $\text{deptDetail}(\text{dId}, \text{dname}) \leftarrow \pi_{\text{deptId}, \text{name}}(\text{department})$
- $\text{profDept} \leftarrow \text{profDetail} \times \text{deptDetail}$
- $\text{desiredProfDept} \leftarrow \sigma_{\text{deptno} = \text{dId}}(\text{profDept})$
- $\text{result} \leftarrow \pi_{\text{eId}, \text{pname}, \text{dname}}(\text{desiredProfDept})$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

28

Query using *cross product* – use of renaming

Query: Obtain the list of professors (Id and Name) along with the *name* of their respective departments

- $\text{profDetail}(\text{eId}, \text{pname}, \text{deptno}) \leftarrow \pi_{\text{empId}, \text{name}, \text{deptNo}}(\text{professor})$
 - this is a temporary relation to hold the intermediate result
 - “empId, name, deptNo” are being renamed as “eId, pname, deptno”
 - creating such relations helps us understand/formulate the query
 - we use “ \leftarrow ” to indicate assignment operation.
- $\text{deptDetail}(\text{dId}, \text{dname}) \leftarrow \pi_{\text{deptId}, \text{name}}(\text{department})$
 - another temporary relation
- Renaming is necessary to ensure that the cross product has distinct attribute names.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

29

Use of renaming operator ρ

Query: Obtain the list of professors (Id and Name) along with the *name* of their respective departments

- One can use the rename operator ρ and write the whole query as one big expression (as an alternative to using temporary relations)

$$\pi_{\text{eId}, \text{pname}, \text{dname}} \left(\left(\sigma_{\text{deptno} = \text{dId}} \left(\rho_{\text{eId}, \text{pname}, \text{deptno}} \left(\pi_{\text{empId}, \text{name}, \text{deptNo}}(\text{professor}) \right) \right) \times \rho_{\text{dId}, \text{dname}} \left(\pi_{\text{deptId}, \text{name}}(\text{department}) \right) \right) \right)$$

- It is easier to understand and formulate the query with *meaningfully named* temporary relations as shown earlier.
- Students are encouraged to use temporary relations.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

30

Join Operation

- **Cross product** : produces all combinations of tuples
 - often only certain combinations are meaningful
 - cross product is usually followed by selection
- **Join** : combines tuples from two relations provided they satisfy a specified condition (join condition)
 - equivalent to performing *cross product* followed by *selection*
 - a very useful operation
- Depending on the type of condition we have
 - *theta join*
 - *equi join*

Prof P Sreenivasa Kumar
Department of CS&E, IITM

31

Theta join

- Let r_1 - relation with scheme $R_1 = (A_1, A_2, \dots, A_m)$
 r_2 - relation with scheme $R_2 = (B_1, B_2, \dots, B_n)$
 where w.l.o.g we assume $R_1 \cap R_2 = \phi$
- Notation for join expression : $r = r_1 \bowtie_{\theta} r_2$
- θ - the join condition - is of the form : $C_1 \wedge C_2 \wedge \dots \wedge C_s$
 C_i is of the form : $A_j \text{ <CompOp> } B_k$
 where <CompOp> is one of $\{ =, \neq, <, \leq, >, \geq \}$
- Scheme of the result relation r is:
 $(A_1, A_2, \dots, A_m, B_1, B_2, \dots, B_n)$
 $r = \{ (a_1, a_2, \dots, a_m, b_1, b_2, \dots, b_n) \mid (a_1, a_2, \dots, a_m) \in r_1, (b_1, b_2, \dots, b_n) \in r_2 \text{ and } (a_1, a_2, \dots, a_m, b_1, b_2, \dots, b_n) \text{ satisfies } \theta \}$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

32

Professor

emplId	name	sex	startYear	deptNo	phone
CS01	GIRIDHAR	M	1984	1	22576345
CS02	KESHAV MURTHY	M	1989	1	22576346
ES01	RAJIV GUPTHA	M	1980	2	22576244
ME01	TAHIR NAYYAR	M	1999	3	22576243

For each department, find its name and the name, sex and phone number of the head of the department

Department

deptId	name	hod	phone
1	Computer Science	CS01	22576235
2	Electrical Engg.	ES01	22576234
3	Mechanical Engg.	ME01	22576233

Courses

courseId	cname	credits	deptNo
CS635	Algorithms	3	1
CS636	A.I	4	1
ES456	D.S.P	3	2
ME650	Aero Dynamics	3	3

Prof P Sreenivasa Kumar
Department of CS&E, IITM

33

Example

For each department, find its name and the name, sex and phone number of the head of the department.

$\text{prof}(\text{empld}, \text{p-name}, \text{sex}, \text{deptNo}, \text{prof-phone})$
 $\leftarrow \pi_{\text{empld}, \text{name}, \text{sex}, \text{deptNo}, \text{phone}}(\text{professor})$
 $\text{result} \leftarrow \pi_{\text{deptId}, \text{name}, \text{hod}, \text{p-name}, \text{sex}, \text{prof-phone}}(\text{department} \bowtie_{(\text{hod} = \text{empld})} \text{prof})$

deptId	name	hod	p-name	sex	prof-phone
1	Computer Science	CS01	Giridher	M	22576235
2	Electrical Engg.	EE01	Rajiv Gupta	M	22576234
3	Mechanical Engg.	ME01	Tahir Nayyar	M	22576233

Prof P Sreenivasa Kumar
Department of CS&E, IITM

34

Equi-join and Natural join

- *Equi-join* : Equality is the only comparison operator used in the join condition
- *Natural join* : R_1, R_2 - have common attributes, say X_1, X_2, X_3
 - Join condition:
 $(R_1.X_1 = R_2.X_1) \wedge (R_1.X_2 = R_2.X_2) \wedge (R_1.X_3 = R_2.X_3)$
 - Values of common attributes should be equal
 - Schema for the result $Q = R_1 \cup R_2 - \{X_1, X_2, X_3\}$
 - Only one copy of the common attributes is kept
- Notation for natural join : $r = r_1 * r_2$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

35

Example – Equi-join

Find courses offered by each department

$\pi_{\text{deptId}, \text{name}, \text{courseId}, \text{cname}, \text{credits}}(\text{Department} \bowtie_{(\text{deptId} = \text{deptNo})} \text{Courses})$

deptId	name	courseId	cname	credits
1	Computer Science	CS635	Algorithms	3
1	Computer Science	CS636	A.I	4
2	Electrical Engg.	ES456	D.S.P	3
3	Mechanical Engg.	ME650	Acro Dynamics	3

Prof P Sreenivasa Kumar
Department of CS&E, IITM

36

Teaching

emplId	courseld	sem	year	classRoom
CS01	CS635	1	2005	BSB361
CS02	CS636	1	2005	BSB632
ES01	ES456	2	2004	ESB650
ME01	ME650	1	2004	MSB331

To find the courses handled by each professor
 Professor * Teaching
 result

emplId	name	sex	startYear	deptNo	phone	courseld	sem	year	classRoom
CS01	Giridhar	M	1984	1	22576345	CS635	1	2005	BSB361
CS02	Keshav Murthy	M	1989	1	22576346	CS636	1	2005	BSB632
ES01	Rajiv Gupta	M	1989	2	22576244	ES456	2	2004	ESB650
ME01	Tahir Nayyar	M	1999	3	22576243	ME650	1	2004	MSB331

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

37

Division operator

- The necessary condition to determine $r \div s$ on instances $r(R)$ and $s(S)$ is $S \subset R$
- The relation $r \div s$ is a relation on schema $R - S$.
A tuple t is in $r \div s$ if and only if
 - t is in $\pi_{R-S}(r)$
 - For every tuple t_s in s , there is t_r in r satisfying both
 - $t_r[S] = t_s$
 - $t_r[R - S] = t$

// $t_r[S]$ – the sub-tuple of t_r consisting of values of attributes in S

- Another Definition $r = r_1 \div r_2$
Division operator produces a relation $R(X)$ that includes all tuples $t[X]$ that appear in r_1 in combination with every tuple from r_2 where $R_1 = Z$ and $R_2 = Y$ and $Z = X \cup Y$

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

38

$R = (A, B, C, D), S = (A, B), X = (C, D)$
 $x = r \div s$

s	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> </tr> </thead> <tbody> <tr> <td>a₁</td> <td>b₁</td> </tr> <tr> <td>a₂</td> <td>b₂</td> </tr> </tbody> </table>	A	B	a ₁	b ₁	a ₂	b ₂	r	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>C</th> <th>D</th> </tr> </thead> <tbody> <tr> <td>a₁</td> <td>b₁</td> <td>c₁</td> <td>d₁</td> </tr> <tr> <td>a₂</td> <td>b₂</td> <td>c₁</td> <td>d₁</td> </tr> <tr> <td>a₁</td> <td>b₁</td> <td>c₂</td> <td>d₂</td> </tr> <tr> <td>a₁</td> <td>b₁</td> <td>c₃</td> <td>d₃</td> </tr> <tr> <td>a₂</td> <td>b₂</td> <td>c₃</td> <td>d₃</td> </tr> </tbody> </table>	A	B	C	D	a ₁	b ₁	c ₁	d ₁	a ₂	b ₂	c ₁	d ₁	a ₁	b ₁	c ₂	d ₂	a ₁	b ₁	c ₃	d ₃	a ₂	b ₂	c ₃	d ₃
A	B																																
a ₁	b ₁																																
a ₂	b ₂																																
A	B	C	D																														
a ₁	b ₁	c ₁	d ₁																														
a ₂	b ₂	c ₁	d ₁																														
a ₁	b ₁	c ₂	d ₂																														
a ₁	b ₁	c ₃	d ₃																														
a ₂	b ₂	c ₃	d ₃																														
x	<table border="1"> <thead> <tr> <th>C</th> <th>D</th> </tr> </thead> <tbody> <tr> <td>c₁</td> <td>d₁</td> </tr> <tr> <td>c₃</td> <td>d₃</td> </tr> </tbody> </table>	C	D	c ₁	d ₁	c ₃	d ₃																										
C	D																																
c ₁	d ₁																																
c ₃	d ₃																																

(c_2, d_2) is not present in the result of division as it does not appear in combination with all the tuples of s in r

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

39

Query using division operation

Find those students who have enrolled for *all* courses offered in the dept of Computer Science.

Step1: Get the course enrollment information for all students
 $\text{studEnroll} \leftarrow \pi_{\text{rollNo, name, courseId}} (\text{student} * \text{enrollment})$

Step2: Get the course Ids of all courses offered by CS dept
 $\text{csCourse} \leftarrow \pi_{\text{courseId}} (\sigma_{\text{dname} = \text{"Computer Science"}} (\text{courses} \bowtie_{\text{deptId} = \text{deptNo}} \text{dept}))$

Result : $\text{studEnroll} \div \text{csCourse}$

Schema

Suppose result of step 1
(we skip roll number for simplicity)

name	courseId
Mahesh	CS635
Mahesh	CS636
Rajesh	CS635
Piyush	CS636
Piyush	CS635
Deepak	ES456
Lalitha	ME650
Mahesh	ME650

result of step 2
csCourse

courseId
CS635
CS636

Let's assume for a moment that student names are unique!

$\text{studEnroll} \div \text{csCourse}$

name
Mahesh
Piyush

Complete Set of Operators

- Are all Relational Algebra operators essential ?
Some operators can be realized through other operators
- What is the minimal set of operators ?
 - The operators $\{\sigma, \pi, \times, \cup, -\}$ constitute a *complete* set of operators
 - Necessary and sufficient set of operators.
 - Intersection – union and difference
 - Join – cross product followed by selection
 - Division – project, cross product and difference

Example Queries

[Schema](#)

Retrieve the list of female PhD students

$\sigma_{\text{degree} = \text{'phD'} \wedge \text{sex} = \text{'F'}}(\text{student})$

Obtain the name and rollNo of all female BTech students

$\pi_{\text{rollNo}, \text{name}}(\sigma_{\text{degree} = \text{'BTech'} \wedge \text{sex} = \text{'F'}}(\text{student}))$

Obtain the rollNo of students who never obtained an 'E' grade

$\pi_{\text{rollNo}}(\sigma_{\text{grade} \neq \text{'E'}}(\text{enrollment}))$

is incorrect!!

(what if some student gets E in one course and A in another?)

$\pi_{\text{rollNo}}(\text{student}) - \pi_{\text{rollNo}}(\sigma_{\text{grade} = \text{'E'}}(\text{enrollment}))$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

43

More Example Queries

Obtain the department Ids for departments with no lady professor

$\pi_{\text{deptId}}(\text{dept}) - \pi_{\text{deptId}}(\sigma_{\text{sex} = \text{'F'}}(\text{professor}))$

Obtain the rollNo of male students who have obtained at least one S grade

$\pi_{\text{rollNo}}(\sigma_{\text{sex} = \text{'M'}}(\text{student})) \cap \pi_{\text{rollNo}}(\sigma_{\text{grade} = \text{'S'}}(\text{enrollment}))$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

44

Another Example Query

[Schema](#)

Obtain the names, roll numbers of students who have got S grade in the CS3700 course offered in 2017 odd semester along with his/her advisor name.

reqStudsRollNo \leftarrow

$\pi_{\text{rollNo}}(\sigma_{\text{courseId} = \text{'CS3700'} \wedge \text{year} = \text{'2017'} \wedge \text{sem} = \text{'odd'} \wedge \text{grade} = \text{'S'}}(\text{enrollment}))$

reqStuds-Name-AdvId (rollNo, sName, advId) \leftarrow

$\pi_{\text{rollNo}, \text{name}, \text{advisor}}(\text{reqStudsRollNo} * \text{student})$

result(rollNo, studentName, advisorName) \leftarrow

$\pi_{\text{rollNo}, \text{sName}, \text{name}}(\text{reqStuds-Name-AdvId} \bowtie_{\text{advId} = \text{empId}} \text{professor})$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

45

Transitive Closure Queries

Schema

Obtain the courses that are either direct or indirect prerequisites of the course CS767.

- Indirect prerequisite – (prerequisite of)⁺ a prerequisite course
- Prerequisites at all levels are to be reported

$\text{levelOnePrereq}(\text{cld1}) \leftarrow \pi_{\text{preReqCourse}}(\sigma_{\text{courseId} = \text{'CS767'}}(\text{preRequisite}))$

$\text{levelTwoPrereq}(\text{cld2}) \leftarrow \pi_{\text{preReqCourse}}(\text{preRequisite} \bowtie_{\text{courseId} = \text{cld1}} \text{levelOnePrereq})$

Similarly, level k prerequisites can be obtained.

But, prerequisites at all levels can not be obtained as there is no looping mechanism.


Prof P Sreenivasa Kumar
Department of CS&E, IITM


46


Outer Join Operation (1/2)

- Theta join, equi-join, natural join are all called *inner joins*. The result of these operations contain only the matching tuples
- The set of operations called *outer joins* are used when all tuples in relation r or relation s or both in r and s have to be in result.

There are 3 kinds of outer joins:

Left outer join 

Right outer join 

Full outer join 

Prof P Sreenivasa Kumar
Department of CS&E, IITM

47

Outer Join Operation (2/2)

Left outer join: $r \bowtie_{\text{left}} s$

It keeps all tuples in the first, or left relation r in the result. For some tuple t in r , if no matching tuple is found in s then S-attributes of t are made null in the result.

Right outer join: $r \bowtie_{\text{right}} s$

Same as above but tuples in the second relation are all kept in the result. If necessary, R-attributes are made null.

Full outer join: $r \bowtie_{\text{full}} s$

All the tuples in both the relations r and s are in the result.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

48

Instance Data for Examples

Student

rollNo	name	degree	year	sex	deptNo	advisor
CS04S001	Mahesh	M.S	2004	M	1	CS01
CS05S001	Amrish	M.S	2003	M	1	null
CS04M002	Piyush	M.E	2004	M	1	CS01
ES04M001	Deepak	M.E	2004	M	2	null
ME04M001	Lalitha	M.E	2004	F	3	ME01
ME03M002	Mahesh	M.S	2003	M	3	ME01

Professor

empld	name	sex	startYear	deptNo	phone
CS01	GIRIDHAR	M	1984	1	22576345
CS02	KESHAV MURTHY	M	1989	1	22576346
ES01	RAJIV GUPTHA	M	1980	2	22576244
ME01	TAHIR NAYYAR	M	1999	3	22576243

Prof P Sreenivasa Kumar
Department of CS&E, IITM

49

Left outer join

temp \leftarrow (student $\bowtie_{\text{advisor} = \text{empld}}$ professor)

$\rho_{\text{rollNo, name, advisor}} (\pi_{\text{rollNo, student.name, professor.name}} (\text{temp}))$

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS05S001	Amrish	Null
CS04M002	Piyush	Giridhar
ES04M001	Deepak	Null
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

Prof P Sreenivasa Kumar
Department of CS&E, IITM

50

Right outer join

temp \leftarrow (student $\bowtie_{\text{advisor} = \text{empld}}$ professor)

$\rho_{\text{rollNo, name, advisor}} (\pi_{\text{rollNo, student.name, professor.name}} (\text{temp}))$

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS04M002	Piyush	Giridhar
null	null	Keshav Murthy
null	null	Rajiv Gupta
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

Prof P Sreenivasa Kumar
Department of CS&E, IITM

51

Full outer join

$\text{temp} \leftarrow (\text{student} \bowtie_{\text{advisor} = \text{empId}} \text{professor})$

$\rho_{\text{rollNo, name, advisor}} (\pi_{\text{rollNo, student.name, professor.name}} (\text{temp}))$

Result

rollNo	name	advisor
CS04S001	Mahesh	Giridhar
CS04M002	Priyush	Giridhar
CS05S001	Amrish	Null
null	null	Keshav Murthy
ES04M001	Deepak	Null
null	null	Rajiv Gupta
ME04M001	Lalitha	Tahir Nayyer
ME03M002	Mahesh	Tahir Nayyer

Prof P Sreenivasa Kumar
Department of CS&E, IITM

52

E/R Diagrams to Relational Schema

- E/R model and the relational model give different representations of a real world enterprise
- An E/R diagram can be converted to a collection of relations
- For each entity set and relationship set in E/R diagram we will have a corresponding relational table with the same name as entity set / relationship set
- Each table will have multiple columns whose names are obtained from the attributes of entity types/relationship types

Prof P Sreenivasa Kumar
Department of CS&E, IITM

53

Relational representation of strong entity sets

- Create a table T_i for each strong entity set E_i .
- Include simple attributes and simple components of composite attributes of entity set E_i as attributes of T_i .
 - Multi-valued attributes of entities are dealt with separately.
- The primary key of E_i will also be the primary key of T_i .
- The primary key can be referred to by other tables via foreign keys in them to capture relationships as we see later

Prof P Sreenivasa Kumar
Department of CS&E, IITM

54

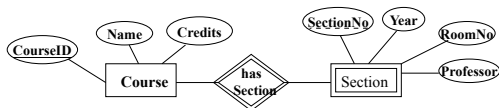
Relational representation of weak entity sets

- Let E' be a weak entity owned by a strong/weak entity E
- E' is converted to a table, say R', where...
- Attributes of R' will be
 - Attributes of the weak entity set E' and
 - Primary key attributes of the identifying strong entity E
(Or, partial key of E + primary key of the owner of E,
if E is itself a weak entity)
 - These attributes will also be a foreign key in R' referring to the table corresponding to E
- Key of R' : partial key of E' + Key of E
- Multi-valued attributes of E' are dealt separately as described later

Prof P Sreenivasa Kumar
Department of CS&E, IITM

55

Example



Corresponding tables are

course			section				
<u>courseId</u>	name	credits	sectionNo	courseId	year	roomNo	professor

Primary key of section = {courseId, sectionNo}

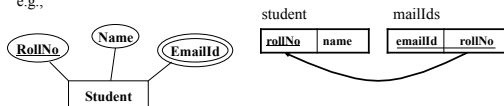
Prof P Sreenivasa Kumar
Department of CS&E, IITM

56

Relational representation of multi-valued attributes

- One separate table for each multi-valued attribute
- One column for this attribute and
- Column(s) for the primary key attribute(s) of the table that corresponds to the entity / relationship set for which this is an attribute.

e.g.,



Prof P Sreenivasa Kumar
Department of CS&E, IITM

57

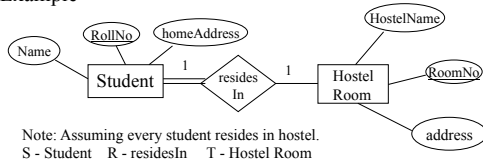
Handling Binary 1:1 Relationship

- Let S and T be entity sets in relationship R and S' and T' be the tables corresponding to these entity sets
- Choose an entity set which has total participation in R, if there is one (say, S)
- Include the primary key of T' as a foreign key in S' referring to relation T'
- Include all simple attributes (and simple components of composite attributes) of R as attributes of S'
- We can do the other way round too
– lot of null values

Prof P Sreenivasa Kumar
Department of CS&E, IITM

58

Example



Student				Hostel		
RollNo	Name	homeAddress	RoomId	RoomNo	HostelName	address

Both entity sets participate fully:
We can merge relations of both
into one "merged" relation.

Foreign key name need
not be same as primary key
of the other relation

Prof P Sreenivasa Kumar
Department of CS&E, IITM

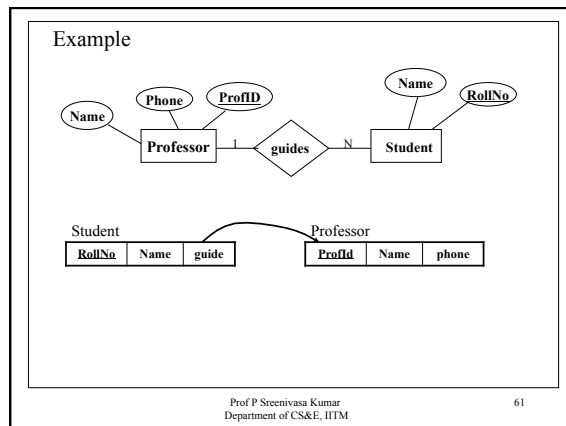
59

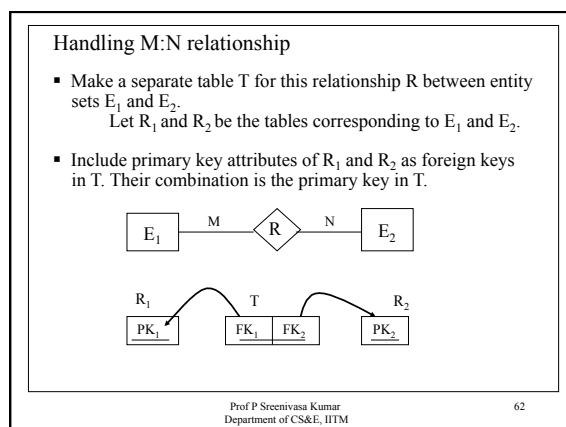
Handling 1: N Relationship

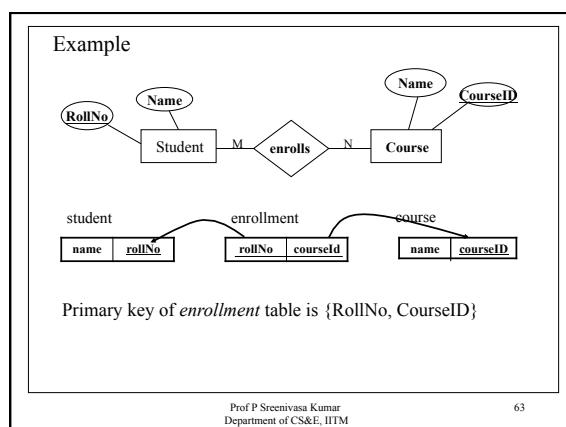
- Let S be the participating entity on the N-side and T the other entity. Let S' and T' be the corresponding tables.
- Include primary key of T' as foreign key in S'
- Include any simple attribute (and simple components of composite attributes) of 1:N relation type as attributes of S'

Prof P Sreenivasa Kumar
Department of CS&E, IITM

60

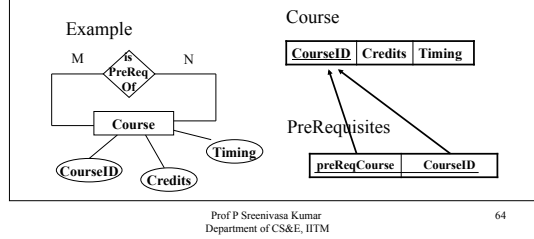






Handling Recursive relationships

- Make a table T for the participating entity set E (this might already be existing) and one table for recursive relationship R.



The SQL Standard

- SQL – Structured Query Language
 - An international standard (ANSI, ISO) that specifies how
 - a relational schema is created
 - data is inserted / updated in the relations
 - data is queried
 - transactions are started and stopped
 - programs access data in the relations
 - and a host of other things are done
- Every relational database management system (RDBMS) is required to support / implement the SQL standard.
 - RDBMS vendors may give additional features
 - Downside of using vendor-specific features - portability

Prof P Sreenivasa Kumar
Department of CS&E, IITM

1

History of SQL

SEQUEL

- developed by IBM in early 70's
- relational query language as part of System-R project at IBM San Jose Research Lab.
- the earliest version of SQL

SQL evolution

- SQL- 86/89
 - SQL- 92 - SQL2
 - SQL- 99/03 - SQL3
- (includes object relational features)

And the evolution continues

Disclaimer: This module covers only important principles of SQL

Prof P Sreenivasa Kumar
Department of CS&E, IITM

2

Components of SQL Standard(1/2)

- *Data Definition Language (DDL)*
Specifies constructs for schema definition, relation definition, integrity constraints, views and schema modification.
- *Data Manipulation Language (DML)*
Specifies constructs for inserting, updating and querying the data in the relational instances (or tables).
- *Embedded SQL and Dynamic SQL*
Specifies how SQL commands can be embedded in a high-level host language such as C, C++ or Java for programmatic access to the data.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

3

Components of SQL Standard(2/2)

- *Transaction Control*
Specifies how transactions can be started / stopped, how a set of concurrently executing transactions can be managed.
- *Authorization*
Specifies how to restrict a user / set of users to access only certain parts of data, perform only certain types of queries etc.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

4

Data Definition in SQL

Defining the schema of a relation

create table *r* (attributeDefinition-1, attributeDefinition-2,...,
name of the relation attributeDefinition-n, [integrityConstraints-1],
[integrityConstraints-2],...,[integrityConstraints-m])

Attribute Definition –

attribute-name domain-type [NOT NULL] [DEFAULT v]

E.g.:

create table *example1* (A char(6) not null default "000000",
B int, C char(1) default "F");

Prof P Sreenivasa Kumar
Department of CS&E, IITM

5

Domain Types in SQL-92 (1/2)

- *Numeric data types*
 - integers of various sizes – INT, SMALLINT
 - real numbers of various precision – REAL, FLOAT, DOUBLE PRECISION
 - formatted numbers – DECIMAL (i, j) or NUMERIC (i, j)
 - i – total number of digits (precision)
 - j – number of digits after the decimal point (scale)
- *Character string data types*
 - fixed length – CHAR(n) – n: no. of characters
 - varying length – VARCHAR(n) – n: max.no. of characters
- *Bit string data types*
 - fixed length – BIT(n)
 - varying length – BIT VARYING(n)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

6

Domain Types in SQL-92 (2/2)

- *Date data type*
DATE type has 10 position format – YYYY-MM-DD
- *Time data type*
TIME type has 8 position format – HH : MM : SS
- *Others*
There are several more data types whose details are available in SQL reference books

Prof P Sreenivasa Kumar
Department of CS&E, IITM

7

Specifying Integrity Constraints in SQL

Also called Table Constraints
Included in the definition of a table

Key constraints

PRIMARY KEY (A_1, A_2, \dots, A_k)
specifies that $\{A_1, A_2, \dots, A_k\}$ is the primary key of the table

UNIQUE (B_1, B_2, \dots, B_k)
specifies that $\{B_1, B_2, \dots, B_k\}$ is a candidate key for the table

There can be more than one UNIQUE constraint but only one PRIMARY KEY constraint for a table.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

8

Specifying Referential Integrity Constraints

FOREIGN KEY (A_1) REFERENCES r_2 (B_1)

- specifies that attribute A_1 of the table being defined, say r_1 , is a *foreign key* referring to attribute B_1 of table r_2
- recall that this means:
each value of column A_1 is either null or is one of the values appearing in column B_1 of r_2

Prof P Sreenivasa Kumar
Department of CS&E, IITM

9

Specifying What to Do if RIC Violation Occurs

RIC violation

- can occur if a referenced tuple is deleted or modified
- action can be specified for each case using qualifiers
ON DELETE or ON UPDATE

Actions

- three possibilities can be specified
SET NULL, SET DEFAULT, CASCADE
- these are actions to be taken on the referencing tuple
- SET NULL – foreign key attribute value to be set null
- SET DEFAULT – foreign key attribute value to be set to its default value
- CASCADE – delete the referencing tuple if the referenced tuple is deleted or update the FK attribute if the referenced tuple is updated

Prof P Sreenivasa Kumar
Department of CS&E, IITM

10

Table Definition Example

```
create table students (
    rollNo char(8) not null,
    name varchar(15) not null,
    degree char(5),
    year smallint,
    sex char not null,
    deptNo smallint,
    advisor char(6),
    primary key(rollNo),
    foreign key(deptNo) references
        department(deptId)
        on delete set null on update cascade,
    foreign key(advisor) references
        professor(empId)
        on delete set null on update cascade
);
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

11

Modifying a Defined Schema

ALTER TABLE command can be used to modify a schema

Adding a new attribute

ALTER table student ADD address varchar(30);

Deleting an attribute

- need to specify what needs to be done about views or constraints that refer to the attribute being dropped
 - two possibilities
 - CASCADE – delete the views/constraints also
 - RESTRICT – do not delete the attributes if there are some views/constraints that refer to it.
 - ALTER TABLE student DROP degree RESTRICT
- Similarly, an entire table definition can be deleted

Prof P Sreenivasa Kumar
Department of CS&E, IITM

12

Data Manipulation in SQL

Basic query syntax

select A_1, A_2, \dots, A_m ← a set of attributes
from R_1, R_2, \dots, R_p ← from relations R_1, \dots, R_p that are
where θ ← the set of tables that
← a boolean predicate that contain the relevant
specifies when a combined tuples to answer the query.
tuple of R_1, \dots, R_p contributes
to the output.

Equivalent to:

$\pi_{A_1, A_2, \dots, A_m}(\sigma_{\theta}(R_1 \times R_2 \times \dots \times R_p))$ Assuming that each attribute
name appears exactly once
in the table.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

13

Meaning of the Basic Query Block

- The *cross product* M of the tables in the from clause would be considered.
Tuples in M that satisfy the condition θ are *selected*.
For each such tuple, values for the attributes A_1, A_2, \dots, A_m (mentioned in the select clause) are *projected*.
- This is a conceptual description
- in practice more efficient methods are employed for evaluation.
- The word *select* in SQL should not be confused with select operation of relational algebra.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

14

SQL Query Result

The result of any SQL query

- a table with *select* clause attributes as column names.
- duplicate rows may be present.
- differs from the definition of a relation.
- duplicate rows can be eliminated by specifying DISTINCT keyword in the *select* clause, if necessary.

SELECT DISTINCT name
FROM student WHERE ...
- duplicate rows are essential while computing aggregate functions (average, sum etc).
- removing duplicate rows involves additional effort and is done only when necessary.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

15

Example Relational Scheme with RICs shown

student (rollNo, name, degree, year, sex, deptNo, advisor)

department (deptId, name, hod, phone)

professor (empId, name, sex, startYear, deptNo, phone)

course (courseId, cname, credits, deptNo)

enrollment (rollNo, courseId, sem, year, grade)

teaching (empId, courseId, sem, year, classRoom)

preRequisite (preReqCourse, courseId)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

16

Example Queries Involving a Single Table

Get the rollNo, name of all women students in the dept no. 5.

```
select rollNo, name
from student
where sex = 'F' and deptNo = 5;
```

Get the employee Id, name and phone number of professors in the CS dept (deptNo = 3) who have joined after 1999.

```
select empId, name, phone
from professor
where deptNo = 3 and startYear > 1999;
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

17

Examples Involving Two or More Relations (1/2)

Get the rollNo, name of students in the CSE dept (deptNo = 3) along with their advisor's name and phone number.

```
select rollNo, s.name, f.name as advisorName,
phone as advisorPhone
from student as s, professor as f
where s.advisor = f.empId and
s.deptNo = 3;
```

attribute renaming in the output

table aliases are required if an attribute name appears in more than one table. Also when same relation appears twice in the from clause.

table aliases are used to disambiguate the common attributes

Prof P Sreenivasa Kumar
Department of CS&E, IITM

18

Examples Involving Two or More Relations (2/2)

Get the names, employee ID's, phone numbers of professors in CSE dept who joined before 1995.

```
select empId, f.name, f.phone
from professor as f, department as d
where f.deptNo = d.deptId and
d.name = 'CSE' and
f.startYear < 1995
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

19

Nested Queries or Sub-queries

While dealing with certain complex queries

- beneficial to specify part of the computation/requirement as a separate query and make use of its result to formulate the main query.
- such queries – nested / sub-queries.

Using sub-queries

- makes the main query easy to understand / formulate
- sometimes makes it more efficient also
 - sub query result can be computed once and used many times.
 - not the case with all sub-queries.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

20

Nested Query Example

Get the rollNo, name of students who have a lady professor as their advisor.

```
select s.rollNo, s.name
from student s
where s.advisor IN
(select empId
from professor
where sex = 'F');
```

IN Operator: One of the ways of making use of the subquery result

Subquery computes the empId's of lady professors

NOT IN can be used in the above query to get details of students who do not have a lady professor as their advisor.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

21

Set Comparison Operators

SQL supports several operators to deal with subquery results or in general with collection of tuples.

Combination of { =, <, ≤, ≥, >, <> } with keywords { ANY, ALL } can be used as set comparison operators.

Get the empId, name of the senior-most Professor(s):

```
select p.empId, p.name
from professors p
where p.startYear <= ALL ( select distinct startYear
                           from professor );
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

22

Semantics of Set Comparison Operators

- νop ANY S

op is one of $<, \leq, >, \geq, =, < >$

true if for some member x of S , $\nu op x$ is true

false if for no member x of S , $\nu op x$ is true
- νop ALL S

S is a subquery

true if for every member x of S , $\nu op x$ is true

false if for some member x of S , $\nu op x$ is not true
- IN is equivalent to $= ANY$
 $NOT IN$ is equivalent to $< > ALL$
- ν is normally a single attribute, but while using IN or $NOT IN$ it can be a tuple of attributes

Prof P Sreenivasa Kumar
Department of CS&E, IITM

23

Correlated and Uncorrelated Nested Queries

If the nested query result is independent of the current tuple being examined in the outer query, nested query is called *uncorrelated*, otherwise, nested query is called *correlated*.

Uncorrelated nested query

- nested query needs to be computed only once.

Correlated nested query

- nested query needs to be re-computed for each row examined in the outer query.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

24

Example of a Correlated Subquery

Get the roll number and name of students whose gender is same as their advisor's.

```
select s.rollNo, s.name
from student s
where s.sex = ALL ( select f.sex
                    from professor f
                    where f.empId = s.advisor );
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

25

The *EXISTS* Operator

Using *EXISTS*, we can check if a subquery result is non-empty

EXISTS(S) is *true* if *S* has at least one tuple / member
is *false* if *S* contain no tuples

Get the employee Id and name of professors who advise at least one women student.

```
select f.empId, f.name
from professors f
where EXISTS ( select s.rollNo
               from student s
               where s.advisor = f.empId and
                     s.sex = 'F' );
```

a correlated
subquery

SQL does not have an operator for universal quantification.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

26

The *NOT EXISTS* Operator

Obtain the department Id and name of departments that do not offer any 4 credit courses.

```
select d.deptId, d.name
from department d
where NOT EXISTS ( select courseId
                   from course c
                   where c.deptNo = d.deptId and
                         c.credits = 4 );
```

a correlated
subquery

Queries with *existentially* quantified predicates can be easily specified using *EXISTS* operator.

Queries with *universally* quantified predicates can only be specified after translating them to use *existential* quantifiers.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

27

Example Involving Universal Quantifier

Determine the students who are enrolled for every course taught by Prof Ramanujam. Assume that Prof Ramanujam teaches at least one course.

As SQL does not have universal quantifier, we will rewrite the query this way:

Determine the student(s) who are such that there **does not** exist a course taught by Prof Ramanujam which is **not** enrolled by the student.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

28

Same query expressed in TRC

Determine the students who are enrolled for **every** course taught by Prof Ramanujam. Assume that Prof Ramanujam teaches at least one course.

```

1. {s.rollNo | student (s) ^
2.   (∀c)(course (c) ^
3.     (∃t)(∃p)( ( teaching(t) ^ professor(p) ^
4.       t.courseId = c.courseId ^
5.       p.name = "Ramanujam" ^
6.       p.empId = t.empId ) →
7.       (∃e) (enrollment(e) ^
8.         e.courseId = c.courseId ^
9.         e.rollNo = s.rollNo ^
10.        e.sem = t.sem ^
11.        e.year = t.year
12.        ) )
13.   )
14. }
```

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

29

Query expressed in SQL

```

select s.rollNo, s.name
from student s
where not exists ( select * from course c, professor p
                  where p.name = "Ramanujam" and
                  exists (select * from teaching t1
                        where t1.courseId = c.courseId and
                        t1.empId = p.empId )
                  and not exists
                  ( select * from teaching t2, enrolment e
                    where t2.empId = p.empId and
                    t2.courseId = c.courseId and
                    t2.courseId = e.courseId and
                    t2.sem = e.sem and t2.year = e.year and
                    e.rollNo = s.rollNo)
                  );
```

c is taught
by Prof
Ramanujam

s has not
enrolled for
c when it
was taught
by Prof
Ramanujam

Prof P Sreenivasa Kumar
Department of CS&E, IITM

30

Another Example Involving the Universal Quantifier

Determine the students who have obtained either S or A grade in all the pre-requisite courses of the course CS7890. It is known that CS7890 has at least one pre-requisite.

```
select s.rollNo, s.name
from student s
where NOT EXISTS (select *
                  from preRequisite p
                  where p.courseId = "CS7890" and
                  NOT EXISTS
                  (select *
                   from enrollment e
                   where e.courseId = p.preReqcourse
                   and e.rollNo = s.rollNo and
                   (e.grade = "S" or e.grade = "A")
                  );
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

31

Missing where Clause

If the *where* clause in an SQL query is not specified, it is treated as - the where condition is true for all tuple combinations.

- Essentially no filtering is done on the cross product of from clause tables.

Get the name and contact phone of all Departments.

```
select name, phone
from department
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

32

Union, Intersection and Difference Operations

- In SQL, using operators *UNION*, *INTERSECT* and *EXCEPT*, one can perform set *union*, *intersection* and *difference* respectively.
- Results of these operators are *sets* – i.e duplicates are automatically removed.
- Operands need to be union compatible and also have *same* attribute names in the *same* order.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

33

Example using UNION

Obtain the roll numbers of students who are enrolled for either CS2300 or CS2320 in 2019 odd semester.

```
(SELECT rollNo
  FROM enrollment
 WHERE courseId = 'CS2300' and
        sem = 'odd' and year = '2019' ) UNION
(SELECT rollNo
  FROM enrollment
 WHERE courseId = 'CS2320' and
        sem = 'odd' and year = '2019' );
```

Equivalent to:

```
(SELECT rollNo
  FROM enrollment
 WHERE (courseId = 'CS2300' or courseId = 'CS2320')
        and sem = 'odd' and year = '2019' )
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

34

Example using INTERSECTION

Obtain the roll numbers of students who are enrolled for both CS2300 and CS2320 in 2019 odd semester.

```
(select rollNo
  from enrollment
 where courseId = 'CS2300' and
        sem = 'odd' and year = '2019' )
```

INTERSECT

```
(select rollNo
  from enrollment
 where courseId = 'CS2320' and
        sem = 'odd' and year = '2019' );
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

35

Example using EXCEPT

Obtain the roll numbers of students who are not enrolled for CS2300 course in 2019 odd semester.

```
(SELECT rollNo
  FROM enrollment
 WHERE sem = 'odd' and year = '2019')
EXCEPT
(SELECT rollNo
  FROM enrollment
 WHERE courseId = 'CS2300' and
        sem = 'odd' and year = '2019');
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

36

Aggregation of Data

Data analysis

- to get info on summary and trends in certain attributes
- need for computing aggregate values for data
- total value, average value etc

Aggregate functions in SQL

- five aggregate function are provided in SQL
- AVG, SUM, COUNT, MAX, MIN
- can be applied to any column of a table
- can be used in the *select* clause of SQL queries

Prof P Sreenivasa Kumar
Department of CS&E, IITM

37

Aggregate functions

- **AVG ([DISTINCT]A):**
computes the average of (distinct) values in column A
- **SUM ([DISTINCT]A):**
computes the sum of (distinct) values in column A
- **COUNT ([DISTINCT]A):**
computes the number of (distinct) values in column A or no. of tuples in result
- **MAX (A):** computes the maximum of values in column A
- **MIN (A):** computes the minimum of values in column A

Optional
keyword

Prof P Sreenivasa Kumar
Department of CS&E, IITM

38

Examples involving aggregate functions (1/2)

Suppose data about GATE exam in a particular year is available as a table with schema

gateMarks(*regNo*, *name*, *sex*, *branch*, *city*, *state*, *marks*)

Obtain the total number of students who have taken GATE in CS and their average marks

```
select count(regNo) as CsTotal, avg(marks) as CSAvg
from gateMarks
where branch = 'CS'
```

Output	
CSTotal	CSAvg

Get the maximum, minimum and average marks obtained by Students from the city of Hyderabad

```
select max(marks), min(marks), avg(marks)
from gateMarks
where city = 'Hyderabad';
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

39

Examples involving aggregate functions (2/2)

Get the names of students who obtained the maximum marks in the branch of EC

```
select name, max(marks)
from gateMarks
where branch = 'EC'
```

} Will not work

Only aggregate functions can be specified here. It does not make sense to include normal attributes ! (unless they are grouping attributes – to be seen later)

```
select regNo, name, marks
from gateMarks
where branch = 'EC' and marks = ANY
(select max(marks)
from gateMarks
where branch = 'EC');
```

Correct way of specifying the query

Prof P Sreenivasa Kumar
Department of CS&E, IITM

40

Data Aggregation and Grouping

Grouping

- Partition the set of tuples in a relation into groups based on certain criteria and compute aggregate functions for each group
- All tuples that agree on a set of attributes (i.e have the same value for each of these attributes) are put into a group
- The specified aggregate functions are computed for each group
- Each group contributes one tuple to the output
- All the grouping attributes *must* also appear in the select clause
 - the result tuple of the group is listed along with the values of the grouping attributes of the group

Called the grouping attributes

Prof P Sreenivasa Kumar
Department of CS&E, IITM

41

Examples involving grouping(1/2)

Determine the maximum of the GATE CS marks obtained by students in each city, for all Cities. Assume 4 cities exist - Hyderabad, Chennai, Mysore and Bangalore.

```
select city, max(marks) as maxMarks
from gateMarks
where branch = 'CS'
group by city;
```

Result:

City	maxMarks
Hyderabad	87
Chennai	88
Mysore	90
Bangalore	86

Prof P Sreenivasa Kumar
Department of CS&E, IITM

42

Examples involving grouping(2/2)

In the University database, for each department, obtain the name, deptId and the total number of four credit courses offered by the department

```
select deptId, name, count(*) as totalCourses
from department, course
where deptId = deptNo and credits = 4
group by deptId, name;
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

43

Having clause

After performing grouping, is it possible to report information about only a subset of the groups ?

- Yes, with the help of *having clause* which is always used in conjunction with Group By clause

Report the total enrollment in each course in the even semester of 2014; include only the courses with a minimum enrollment of 10.

```
select courseId, count(rollNo) as Enrollment
from enrollment
where sem = even and year = 2014
group by courseId
having count(rollNo) ≥ 10;
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

44

Where clause versus Having clause

- Where clause
 - Performs tests on rows and eliminates rows not satisfying the specified condition
 - Performed *before* any grouping of rows is done
- Having clause
 - Always performed *after* grouping
 - Performs tests on groups and eliminates groups not satisfying the specified condition
 - Tests can only involve grouping attributes and aggregate functions

```
select courseId, count(rollNo) as Enrollment
from enrollment
where sem = 2 and year = 2014
group by courseId
having count(rollNo) ≥ 10;
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

45

String Operators in SQL

- Specify strings by enclosing them in single quotes
e.g., 'Chennai'

Common operations on strings –

- Pattern matching – using 'LIKE' comparison operator
 - specify patterns using special characters –
- Character '%' (percent) matches any Substring
e.g., 'Ram%' matches any string starting with "Ram"
- Character '_' (underscore) matches any single character
e.g., (a) '____nagar' matches with any string ending with "nagar", with any 3 characters before that.
(b) '_____' matches any string with exactly four characters

Prof P Sreenivasa Kumar
Department of CS&E, IITM

46

Using the 'LIKE' operator

Obtain roll numbers and names of all students whose names end with 'Mohan'

```
select rollNo, name
from student
where name like '%Mohan';
```

- Patterns are case sensitive.
- Special characters (percent, underscore) can be included in patterns using an escape character '\' (backslash)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

47

Join Operation

In SQL, usually joining of tuples from different relations is implicitly specified in the 'where' clause

Get the names of professors working in CSE dept.

```
select f.name
from professor as f, department as d
where f.deptNo = d.deptId and
      d.name = 'CSE';
```

The above query specifies joining of professor and department relations on condition $f.deptNo = d.deptId$ and selection on department relation using $d.name = 'CSE'$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

48

Explicit Specification of Joining in 'From' Clause

```
select f.name
from (professor as f join department as d on
      f.deptNo = d.deptId)
where d.name = 'CSE';
```

Join types:

1. inner join (default):
from (r_1 inner join r_2 on <predicate>)
use of just 'join' is equivalent to 'inner join'
2. left outer join:
from (r_1 left outer join r_2 on <predicate>)
3. right outer join:
from (r_1 right outer join r_2 on <predicate>)
4. full outer join:
from (r_1 full outer join r_2 on <predicate>)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

49

Natural join

The adjective 'natural' can be used with any of the join types to specify natural join.

FROM (r_1 NATURAL <join type> r_2 [USING <attr. list>])

- natural join by default considers all common attributes
- a subset of common attributes can be specified in an optional USING <attr. list> phrase

REMARKS

- Specifying join operation explicitly goes against the spirit of declarative style of query specification
- But the queries may be easier to understand
- The feature is to be used judiciously

Prof P Sreenivasa Kumar
Department of CS&E, IITM

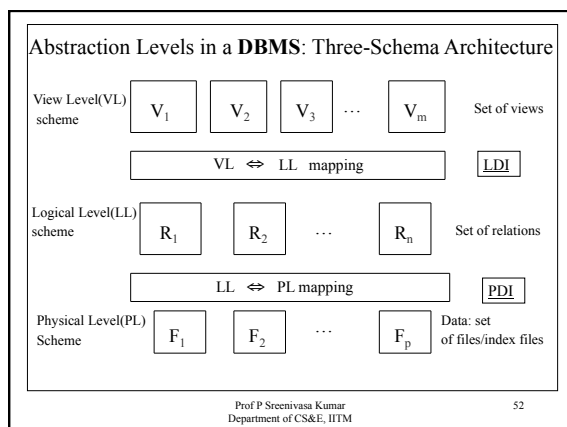
50

Views (or Virtual Tables)

- Views provide virtual relations which may contain data spread across different tables. Used by applications.
 - simplified query formulations
 - data hiding
 - a view of frequently used data – efficient query answering
- Once created, a view is always kept *up-to-date* by the RDBMS
- View is not part of conceptual schema
 - created to give a user group, concerned with a certain aspect of the information system, their *view* of the system
- View implementation
 - Views need not be stored as permanent tables
 - They can be created on-the-fly whenever needed or
 - They can also be *materialized* and kept up-to-date
- Tables involved in the view definition – base tables

Prof P Sreenivasa Kumar
Department of CS&E, IITM

51



Creating Views

CREATE VIEW v AS <query expr>
creates a view 'v', with structure and data defined by the outcome of the query expression

Create a view which contains name, employee Id and phone number of professors who joined CSE dept in or after the year 2000. _____ name of the view

```
create view profAft2K as
(select f.name, empId, phone
 from professor as f, department as d
 where f.depNo = d.deptId and
       d.name = 'CSE' and
       f.startYear >= 2000);
```

If the details of a new CSE professor are entered into *professor* table, the above view gets updated automatically

Prof P Sreenivasa Kumar
Department of CS&E, IITM

Queries on Views

Once created a view can be used in queries just like any other table.

e.g. Obtain names of professors in CSE dept, who joined after 2000 and whose name starts with 'Ram'

```
select name
from profAft2K
where name like 'Ram%';
```

The definition of the view is stored in DBMS, and executed to create the temporary table (view), when encountered in query

Prof P Sreenivasa Kumar
Department of CS&E, IITM

Operations on Views

- Querying is allowed
- Update operations are usually restricted
 - because – to update a view, we may require
 - to modify many base tables
 - there may not be a unique way of updating the base tables to reflect the update on view
 - views may contain some aggregate values
 - ambiguity where primary key of a base table is not included in view definition.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

55

Restrictions on Updating Views

- Updates on views defined on joining of more than one table are not allowed
- For example, updates on the following view are not allowed
- Note that we are not keeping information about when a student has completed the course in the view

create a view StudentGrades with rollNo, name, courseId and grade

```
create view studentGrade(rollNo,name,courseId,grade) as
(select s.rollNo, s.name, e.courseId, e.grade
 from student s, enrollment e
 where s.rollNo = e.rollNo);
```

- Suppose we want to update grade in the view from “U” to “D” for one particular course for a student, there will be ambiguity in doing the update on base tables.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

56

Restrictions on Updating Views

- Updates on views defined with ‘group by’ clause and aggregate functions is not permitted, as a tuple in view will not have a corresponding tuple in base relation.
- For example, updates on the following view are not allowed

Create a view deptAvgCredits which contains the average credits of courses offered by a dept.

```
create view deptAvgCredits(deptNo,avgCredits)
as select deptNo, avg(credits)
 from course
 group by deptNo;
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

57

Restrictions on Updating Views

- Updates on views which do not include Primary Key of base table, are also not permitted
- For example, updates on the following view are not allowed

Create a view StudentPhone with Student name and phone number.

```
create view StudentPhone (sname,sphone) as
(select name, phone
 from student);
```

View StudentPhone does not include Primary key of the base table.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

58

Allowed Updates on Views

Updates to views are allowed only if

- defined on single base table
- not defined using 'group by' clause and aggregate functions
- view includes Primary Key of base table

Prof P Sreenivasa Kumar
Department of CS&E, IITM

59

Inserting data into a table

- Specify a tuple(or tuples) to be inserted
INSERT INTO student VALUES
('CS05D014', 'Mohan', 'Phd', 2005, 'M', 3, 'FCS008'),
('CS05S031', 'Madhav', 'MS', 2005, 'M', 4, 'FCE009');
- Specify the result of query to be inserted
INSERT INTO r₁ SELECT ... FROM ... WHERE ...
- Specify that a sub-tuple be inserted
INSERT INTO student(rollNo, name, sex)
VALUES (CS05M022, 'Rajasri', 'F'),
(CS05B033, 'Kalyan', 'M');
 - the attributes that can be NULL or have declared default values can be left-out to be updated later

Prof P Sreenivasa Kumar
Department of CS&E, IITM

60

Deleting rows from a table

- Deletion of tuples is possible ; deleting only part of a tuple is not possible
- Deletion of tuples can be done *only from one* relation at a time
- Deleting a tuple might trigger further deletions due to *referentially triggered actions* specified as part of RIC's
- Generic form: *delete from r where <predicate>;*

Delete tuples from professor relation with start year as 1982.

```
delete from professor
where startYear = 1982;
```

- If 'where' clause is not specified, then all the tuples of that relation are deleted (Be careful !)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

61

A Remark on Deletion

- The where predicate is evaluated for each of the tuples in the relation to mark them as qualified for deletion *before* any tuple is actually deleted from the relation
- Note that the result may be different if tuples are deleted as and when we find that they satisfy the where condition!
- An example:
Delete all tuples of students that scored the least marks in the CS branch:
DELETE
FROM gateMarks
WHERE branch = "CS" and
marks = ANY (SELECT MIN(marks)
FROM gateMarks
WHERE branch = "CS")

Prof P Sreenivasa Kumar
Department of CS&E, IITM

62

Updating tuples in a relation

```
update r
set <<attr = newValue> list>
where <predicates>;
```

Change phone number of all professors working in CSE dept to "94445 22605"

```
update professors
set phone = '9444422605'
where deptNo = (select deptId
from department
where name = 'CSE');
```

If 'where' clause is not specified, values for the specified attributes in all tuples is changed.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

63

Miscellaneous features in SQL (1/3)

- Ordering of result tuples can be done using 'order by' clause
e.g., List the names of professors who joined after 1980, in alphabetic order.

```
select name
from professor
where startYear > 1980
order by name;
```
- Use of 'null' to test for a null value, if the attribute can take null
e.g., Obtain roll numbers of students who don't have phone numbers

```
select rollNo
from student
where phoneNumber is null;
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

64

Miscellaneous features in SQL (2/3)

- Use of 'between and' to test the range of a value
e.g., Obtain names of professors who have joined between 1980 and 1990

```
select name
from professor
where startYear between 1980 and 1990;
```
- Change the column name in result relation
e.g.,

```
select name as studentName, rollNo as studentNo
from student;
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

65

Miscellaneous features in SQL (3/3)

- Use of 'distinct' key word in 'select' clause to determine duplicate tuples in result.
Obtain all distinct branches of study for students

```
select distinct d.name
from student as s, department as d
where s.deptNo = d.deptId;
```
- Use of asterisk (*) to retrieve all the attribute values of selected tuples.
Obtain details of professors along with their department details.

```
select *
from professor as f, department as d
where f.deptNo = d.deptId;
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

66

Database System Architectures

Centralized Architecture – used long ago, before PCs were born
 Complete DB functionality – storage, running application programs, transaction processing etc – is on one system - Server
 Access systems are just display devices - terminals

Client/Server Architecture – two tier systems

Client – powerful enough to do local processing
 – runs graphical user interface and application programs
 – sends Database queries/updates to Server
 Server – provides rest of the DB System functionality

Three Tier System Architectures – also possible – details left out here

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

67

Application Development Process: 2-tier Systems

Host language (HL) – the high-level programming language in which the application is developed (e.g., C, C++, Java etc.)

Managing Database Access – several approaches are available

- Embedded SQL approach – SQL commands are embedded in the HL programs
 - A static approach - SQL commands can't be given at runtime
 - Dynamic SQL
- Call Level Interface SQL/CLI – an API based approach
- JDBC – Java DB connectivity – an API based approach
- Use a Database programming language – Oracle's PL/SQL

Embedded SQL, Dynamic SQL – we will study in some detail
 Other approaches – to be studied by students

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

68

Impedance Mismatch

▪ Impedance Mismatch:

Problems due to difference in HL data model vs DB data model

- Data types of HL vs those in DB
- HL languages do not support set-of-records as supported by SQL

▪ Handling Data types

- For each SQL attribute data type – corresponding HL data type
- Specified as language *binding*
- To be done for each host language

▪ Handling SQL query results

- Results are either sets or multi-sets of tuples
- A data structure to hold results and an iterator are needed

▪ Does not arise in case of dedicated DB programming languages

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

69

Embedded SQL Approach

Host language (HL) – the high-level programming language in which the application is developed (e.g., C, C++, Java etc.)

Embedded SQL approach:

- SQL statements are interspersed in HL program for application development
- Pre-compiler replaces these with suitable library calls
 - Library is supplied by the RDBMS vendor
- SQL commands – identified by special reserved words – EXEC SQL

Data transfer – takes place through specially declared HL variables

Prof P Sreenivasa Kumar
Department of CS&E, IITM

70

Declaring Variables

Variables that need to be used in SQL statements are declared in a special section. These are called *shared* variables.

```
EXEC SQL BEGIN DECLARE SECTION
char rollNo[9];           //HL is C language
char studName[20], degree[6];
int year; char sex;
int deptNo; char advisor[9];
EXEC SQL END DECLARE SECTION
```

Note that schema for student relation is
student(rollNo, name, degree, year, sex, deptNo, advisor)

Use in SQL statements: variable name is prefixed with a colon(:)
e.g., :ROLLNO in an SQL statement refers to rollNo variable
In HL program, shared variables can be used directly w/o colon.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

71

Handling Error Conditions

The HL program needs to know if an SQL statement has executed successfully or otherwise

Special variable called SQLSTATE is used for this purpose
It is a string of 6 characters.

- SQLSTATE is set to appropriate value by the RDBMS run-time system after executing each SQL statement
- Non-zero values indicate errors in execution
 - Different values indicate different types of error situations
- SQLSTATE variable must be declared in the HL program
- HL program needs to check for error situations and handle them appropriately.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

72

Database Connections in Embedded SQL Approach

DB connection

- Needs to be established before accessing the DB in the app pgm
- Specify the particular server and authenticate the application
- Several connections - to access 2 or more DB servers
- Only one connection can be *active* at any time

SQL Commands

```
CONNECT TO <serverName> AS <connName>
AUTHORIZATION <uName, passWd>
```

To change to a different server
SET CONNECTION <connName>

```
DISCONNECT <connName>
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

73

Embedded SQL Statements – An example

Suppose we collect data through user interface into variables
rollNo, studName, degree, year, sex, deptNo, advisor

A row into the student table can be inserted as follows:

```
EXEC SQL INSERT INTO STUDENT
VALUES (:rollNo, :studName, :degree,
       :year, :sex, :deptNo, :advisor);
```

Prof P Sreenivasa Kumar
Department of CS&E, IITM

74

Query result handling and Cursors

- HL languages do not support set-of-records as supported by SQL
- A *cursor* is a mechanism which allows us to retrieve one row at a time from the result of a query
- We can declare a cursor on any SQL query
- Once declared, we use open, fetch, move and close commands to work with cursors
- We usually need a cursor when embedded statement is a SELECT query
- INSERT, DELETE and UPDATE do not need a cursor.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

75

Embedded SQL - Cursors (1/2)

We do not need a cursor if the query results in a single row.

e.g., *EXEC SQL SELECT s.name, s.sex
INTO :name, :sex
FROM student s
WHERE s.rollNo = :rollNo;*

- Result row values name and phone are assigned to HL variables :name and :phone, using INTO clause
- Cursor is not required as the result always contains only one row (*rollNo* is a key for *student* relation)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

76

Embedded SQL - Cursors (2/2)

- If the result contains more than one row, cursor declaration is needed

e.g., *select s.name, s.degree
from student s
where s.sex = 'F';*

- Query results in a collection of rows
- HL program has to deal with set of records.
- The use of 'INTO' will not work here
- We can solve this problem by using a *cursor*.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

77

Declaring a cursor on a query

***declare studInfo cursor for**
select name, degree
from student
where sex = 'F';*

Cursor name

- Command OPEN studInfo; opens the cursor and makes it point to first record
- To read current row of values into HL variables:
FETCH studInfo INTO :name, :degree;
- After executing FETCH statement cursor is pointed to next row by default
- Cursor movement can be optionally controlled by the programmer
- After reading all records we close the cursor using the CLOSE studInfo command.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

78

Dynamic SQL

- Useful for applications to generate and run SQL statements, based on user inputs
- Queries may not be known in advance

```
e.g., char sqlString[ ] = {"select * from student"};
EXEC SQL PREPARE runQ FROM sqlString;
EXEC SQL EXECUTE runQ;
```

- *sqlString* is a C variable that holds user submitted query
 - Typically built by previous statements in the HL program using end-user inputs.
- *runQ* is an SQL variable that holds the SQL statement.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

79

Connecting to Database from HL – Other Approaches

ODBC (Open Database Connectivity), SQL/CLI and JDBC

- accessing database and data is through an API
- many DBMSs can be accessed
- no restriction on number of active connections
- appropriate drivers are required
- steps in accessing data from a HL program
 - select the data source
 - load the appropriate driver dynamically
 - establish the connection
 - authenticate the client
 - work with database
 - close the connection.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

80

A comparison of the Approaches

Embedded SQL Approach

- + queries are part of source code, syntax-check at compile time
- + application programs are easy to understand, readable
- + development is easier
- Any changes to queries : recompilation required
- Complex applications requiring runtime query creation are difficult

API based Approach

- + More flexibility in programming
- + Complex applications can be developed
- Application development is complex, error-prone

DB Language Approach

- + No impedance mismatch
- Programmers need to learn a new language; apps not portable

Prof P Sreenivasa Kumar
Department of CS&E, IITM

81

File Organization and Indexing

The data of a RDB is ultimately stored in disk files

Disks – non-volatile, inexpensive storage for data
– random-access addressable device

Disk space management:

Should Operating System services be used ?

Should RDBMS manage the disk space by itself ?

2nd option is preferred as RDBMS requires complete control over when a block or page in main memory buffer is written to the disk.

This is important for recovering data when system crash occurs

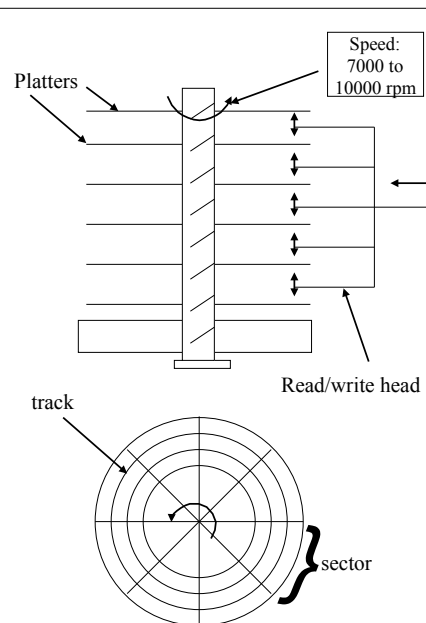
Prof P Sreenivasa Kumar
Department of CS&E, IITM

1

Structure of Disks

Disk

- several platters stacked on a rotating spindle
- one read / write head per surface for fast access
- platter has several tracks
 - ~10,000 per inch
- each track - several sectors
- each sector/track - blocks
- unit of data transfer - block
- cylinder i - track i on all platters
- sectoring is optional
- block – ½ KB to 8KB
 - fixed; set at initialization time



Prof P Sreenivasa Kumar
Department of CS&E, IITM

2

Data Transfer from Disk

Address of a block: Surface No, Cylinder No, Block No

Data transfer:

Move the r/w head to the appropriate track

- time needed - seek time – ~ 12 to 14 ms

Wait for the appropriate block to come under r/w head

- time needed - rotational delay - ~3 to 4ms (avg)

Access time: Seek time + rotational delay

Blocks on the same cylinder - roughly close to each other
- access time-wise

- cylinder i , cylinder $(i + 1)$, cylinder $(i + 2)$ etc.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

3

Data Records and Files

File – a sequence of records

Fixed length record type: each field is of fixed length

- in a file of these type of records, the record number can be used to locate a specific record
- the number of records, the length of each field are available in file header

Variable length record type:

- arise due to missing fields, repeating fields, variable length fields or if different types of records are stored in a file.
- special separator symbols are used to indicate the field boundaries and record boundaries
- the number of records, the separator symbols used, record type codes are all stored in the file header

Prof P Sreenivasa Kumar
Department of CS&E, IITM

4

Packing Records into Blocks

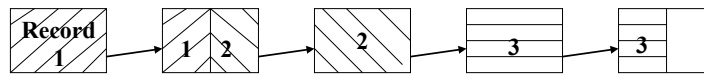
Record length much less than block size

- The usual case
- Blocking factor $b = \lfloor B/r \rfloor$
 - B - block size (bytes)
 - r - record length (bytes)
 - maximum no. of records that can be stored in a block

Un-spanned records are used – a record is not split

Record length greater than block size

- spanned organization is used



File blocks:

sequence of blocks containing all the records of the file

Prof P Sreenivasa Kumar
Department of CS&E, IITM

5

Mapping File Blocks onto the Disk Blocks

Contiguous allocation

- Consecutive file blocks are stored in consecutive disk blocks
- Pros: File scanning can be done fast using double buffering
- Cons: Expanding the file by including a new block in the middle of the sequence - difficult

Linked allocation

- each file block is assigned to some disk block
- each disk block has a pointer to next block of the sequence
- file expansion is easy; but scanning is slow

Mixed allocation - clusters of file blocks are stored consecutively
- clusters are linked in order...

Indexed allocation - index blocks are used.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

6

File Header / File descriptor

Contains information on

- the disk addresses of the file blocks

- record format description

 - field lengths, order of fields

 - for unspanned, fixed-length records

 - field / record separator characters, order of fields, record types

 - for variable length records

Prof P Sreenivasa Kumar
Department of CS&E, IITM

7

Operations on Files

Insertion of a new record: may involve searching for appropriate location for the new record

Deletion of a record: locating a record – may involve search; delete the record – may involve movement of other records

Update a record field/fields: equivalent to delete and insert

Search for a record: given value of a key field / non-key field

Range search: given range values for a key / non-key field

How successfully we can carry out these operations depends on the organization of the file and the availability of indexes

Prof P Sreenivasa Kumar
Department of CS&E, IITM

8

Primary File Organization

The logical policy / method used for placing records into file blocks

Example: *Student* file - organized to have students records sorted in increasing order of the “rollNo” values

Goal: To ensure that operations performed frequently on the file execute fast

- conflicting demands may be there
- example: on student file, access based on rollNo and also access based on name may both be frequent
- we choose to make rollNo access fast
- For making name access fast, additional access structures are needed.
 - more details later

Prof P Sreenivasa Kumar
Department of CS&E, IITM

9

Different File Organization Methods

We will discuss Heap files, Sorted files and Hashed files

Heap file:

Records are appended to the file as they are inserted

Simplest organization

Insertion - Read the last file block, append the record and write back the block - easy

Locating a record given values for any attribute

- requires scanning the entire file – very costly

Heap files are often used only along with other access structures.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

10

Sorted files / Sequential files / Clustered files (1/2)

Ordering field: The field whose values are used for sorting the records in the data file

Ordering key field: An ordering field that is also a key

Sorted file / Sequential file:

Data file whose records are arranged such that the values of the ordering field are in ascending order

Locating a record given the value X of the ordering field:

Binary search can be performed

Address of the n^{th} file block can be obtained from the file header

$O(\log N)$ disk accesses to get the required block- efficient

Range search is also efficient

Prof P Sreenivasa Kumar
Department of CS&E, IITM

11

Sorted files / Sequential files / Clustered files (2/2)

Inserting a new record:

- Ordering gets affected
 - costly as all blocks following the block in which insertion is performed may have to be modified
- Hence not done directly in the file
 - all inserted records are kept in an auxiliary file
 - periodically file is reorganized - auxiliary file and main file are merged
 - locating record
 - carried out first on auxiliary file and then the main file.

Deleting a record

- deletion markers are used.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

12

Hashed Files

Very useful file organization, if quick access to the data record is needed given the value of a single attribute.

Hashing field: The attribute on which quick access is needed and on which hashing is performed

Data file: organized as buckets with numbers $0, 1, \dots, (M - 1)$
(bucket - a block or a few *consecutive* blocks)

Hash function h : maps the values from the domain of the hashing attribute to bucket numbers

Prof P Sreenivasa Kumar
Department of CS&E, IITM

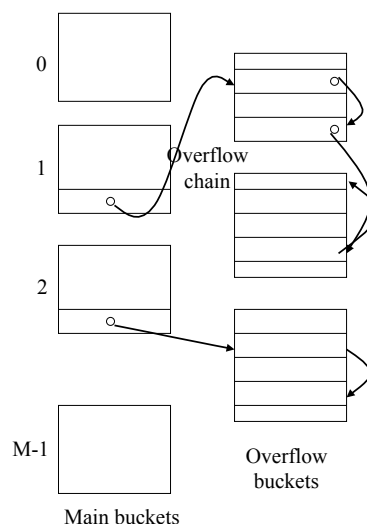
13

Inserting Records into a Hashed File

Insertion: for the given record R , apply h on the value of hashing attribute to get the bucket number r .

If there is space in bucket r , place R there, else place R in the overflow chain of bucket r :

The overflow chains of all the buckets are maintained in the overflow buckets.



Prof P Sreenivasa Kumar
Department of CS&E, IITM

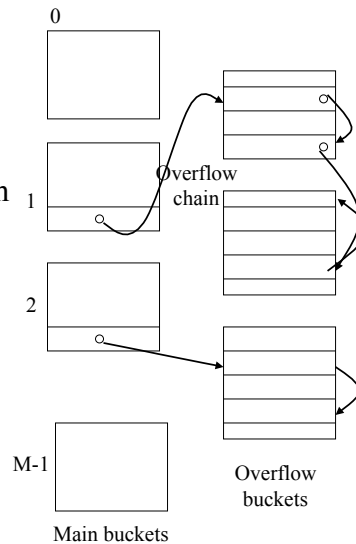
14

Deleting Records from a Hashed File

Deletion: Locate the record R to be deleted by applying h .

Remove R from its bucket/overflow chain. If possible, bring a record from the overflow chain into the bucket

Search: Given the hash filed value k , compute $r = h(k)$. Get the bucket r and search for the record. If not found, search the overflow chain of bucket r :



Prof P Sreenivasa Kumar
Department of CS&E, IITM

15

Performance of Static Hashing

Static hashing:

- The hashing method discussed so far
- The number of main buckets is fixed

Locating a record given the value of the hashing attribute
most often – one block access

Capacity of the hash file $C = r * M$ records
(r - no. of records per bucket, M - no. of main buckets)

Disadvantage with static hashing:

If actual records in the file is much less than C

- wastage of disk space

If actual records in the file is much more than C

- long overflow chains – degraded performance

Prof P Sreenivasa Kumar
Department of CS&E, IITM

16

Hashing for Dynamic File Organization

Dynamic files

- files where record insertions and deletion take place frequently
- the file keeps growing and also shrinking

Hashing for dynamic file organization

- Bucket numbers are integers
- The binary representation of bucket numbers is
 - Exploited cleverly to devise dynamic hashing schemes
 - Two schemes
 - Extendible hashing
 - Linear hashing

Prof P Sreenivasa Kumar
Department of CS&E, IITM

17

Extendible Hashing (1/2)

The k -bit sequence corresponding to a record R :

Apply hashing function to the value of the hashing field of R
to get the bucket number r

Convert r into its binary representation to get the bit sequence
Take the *trailing* k bits

For instance, say record R hashes to bucket # 46

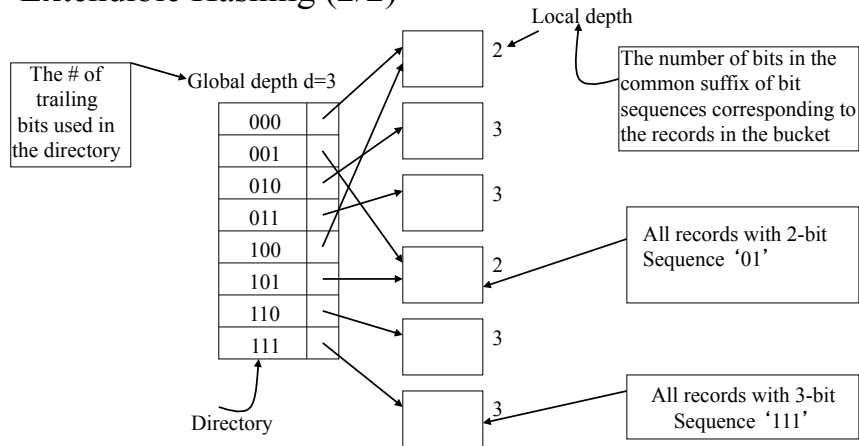
$$46 = (101110)_2$$

So, the 3-bit sequence corresponding to the bucket is “110”

Prof P Sreenivasa Kumar
Department of CS&E, IITM

18

Extendible Hashing (2/2)



Locating a record:

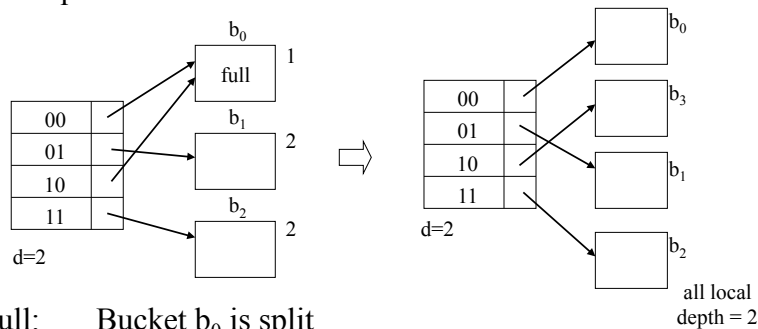
Match the d -bit sequence with an entry in the directory and go to the corresponding bucket to find the record

Prof P Sreenivasa Kumar
Department of CS&E, IITM

19

Insertion in Extendible Hashing Scheme (1/2)

2-bit sequence for the record to be inserted: 00



b_0 Full: Bucket b_0 is split
All records whose 2-bit sequence is '10' are sent to a new bucket b_3 . Others are retained in b_0
Directory is modified.

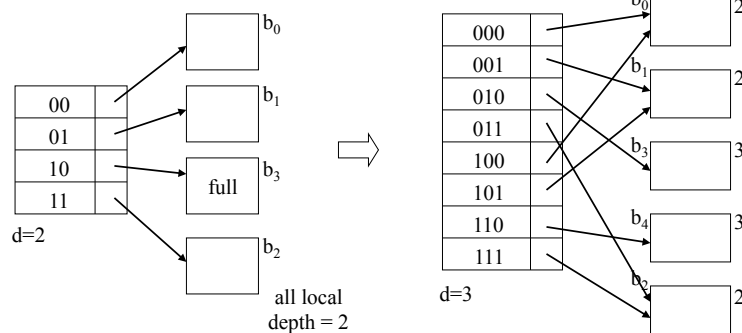
b_0 Not full: New record is placed in b_0 . No changes in the directory.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

20

Insertion in Extendible Hashing Scheme (2/2)

2-bit sequence for the record to be inserted: 10



b_3 not full: new record placed in b_3 . No changes.

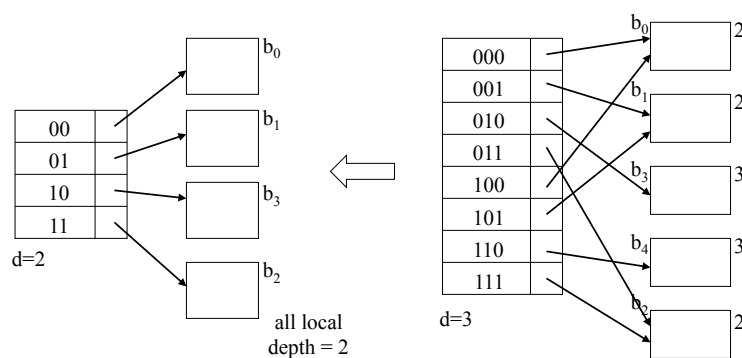
b_3 full : b_3 is split, directory is doubled, all records with 3-bit sequence 110 sent to b_4 . Others in b_3 .

In general, if the local depth of the bucket to be split is equal to the global depth, directory is doubled

Prof P Sreenivasa Kumar
Department of CS&E, IITM

21

Deletion in Extendible Hashing Scheme



Matching pair of data buckets:

k -bit sequences have a common $k-1$ bit suffix, e.g. b_3 & b_4

Due to deletions, if a pair of matching data buckets

-- become less than half full -- try to merge them into one bucket

If the local depth of all buckets is one less than the global depth

-- reduce the directory to half its size

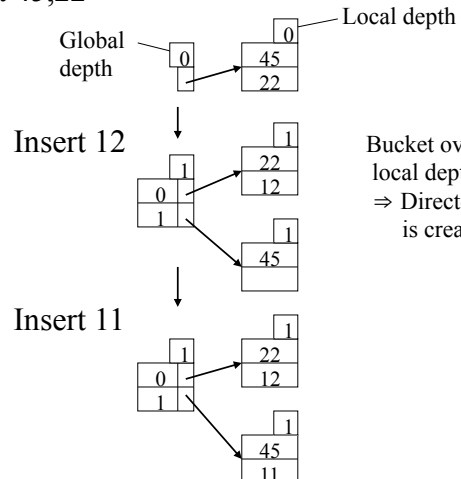
Prof P Sreenivasa Kumar
Department of CS&E, IITM

22

Extendible Hashing Example

Bucket capacity – 2 Initial buckets = 1

Insert 45,22



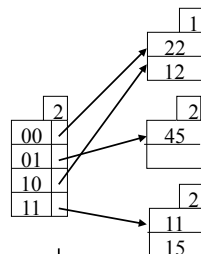
45	101101
22	10110
12	1100
11	1011

Bucket overflows
local depth = global depth
⇒ Directory doubles and split image is created

Prof P Sreenivasa Kumar
Department of CS&E, IITM

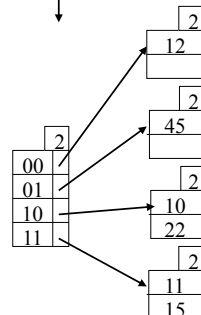
23

Insert 15



Overflow occurs.
Global depth = local depth
Directory doubles and split occurs

Insert 10



Overflows occurs.
Since local depth < global depth
Split image is created
Directory is not doubled

45	101101
22	10110
12	1100
11	1011
15	1111
10	1010

Prof P Sreenivasa Kumar
Department of CS&E, IITM

24

Linear Hashing

Does not require a separate directory structure

Uses a family of hash functions h_0, h_1, h_2, \dots

- the range of h_i is double the range of h_{i-1}
- $h_i(x) = x \bmod 2^i M$
M - the initial no. of buckets
(Assume that the hashing field is an integer)

Initial hash functions

$$h_0(x) = x \bmod M$$

$$h_1(x) = x \bmod 2M$$

Insertion (1/3)

Initially the structure has M main buckets
(0 , ..., M-1) and a few overflow buckets

To insert a record with hash field value x,
place the record in bucket $h_0(x)$

When the *first* overflow in any bucket occurs:

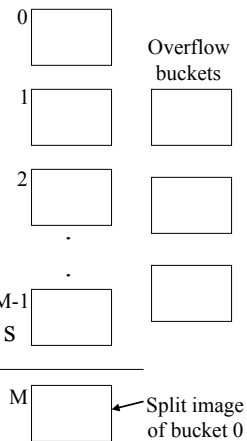
Say, overflow occurred in bucket s

Insert the record in the overflow chain of bucket s

Create a new bucket M

Split the *bucket 0* by using h_1

Some records stay in bucket 0 and
some go to bucket M.



Insertion (2/3)

On first overflow,
irrespective of where it occurs, bucket 0 is split

On subsequent overflows
buckets 1, 2, 3, ... are split in that order

(This why the scheme is called linear hashing)

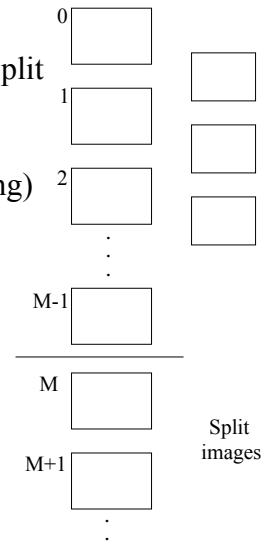
N: the next bucket to be split

After M overflows,

all the original M buckets are split.

We switch to hash functions h_1, h_2
and set $N = 0$.

$h_0 \rightarrow h_1 \rightarrow \dots h_i \rightarrow \dots$
 $h_1 \rightarrow h_2 \rightarrow \dots h_{i+1} \rightarrow \dots$



Prof P Sreenivasa Kumar
Department of CS&E, IITM

27

Nature of Hash Functions

$$h_i(x) = x \bmod 2^i M. \text{ Let } M' = 2^i M$$

- Note that if $h_i(x) = k$ then $x = M'r + k$, $k < M'$

$$\text{and } h_{i+1}(x) = (M'r + k) \bmod 2M'$$

$$= k \text{ or } M' + k$$

Since,

$$r - \text{even} - (M'2s + k) \bmod 2M' = k$$

$$r - \text{odd} - (M'(2s + 1) + k) \bmod 2M' = M' + k$$

M' – the current number of original buckets.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

28

Insertion (3/3)

Say the hash functions in use are h_i, h_{i+1}

To insert record with hash field value x ,

Compute $h_i(x)$

if $h_i(x) < N$, the original bucket is already split

place the record in bucket $h_{i+1}(x)$

else place the record in bucket $h_i(x)$

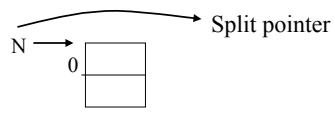
Prof P Sreenivasa Kumar
Department of CS&E, IITM

29

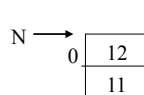
Linear Hashing Example

Initial Buckets = 1 Bucket capacity = 2 records

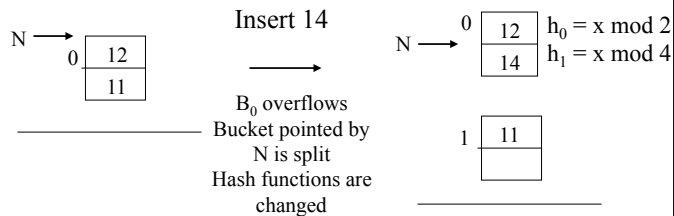
Hash functions
 $h_0 = x \bmod 1$
 $h_1 = x \bmod 2$



Insert 12, 11

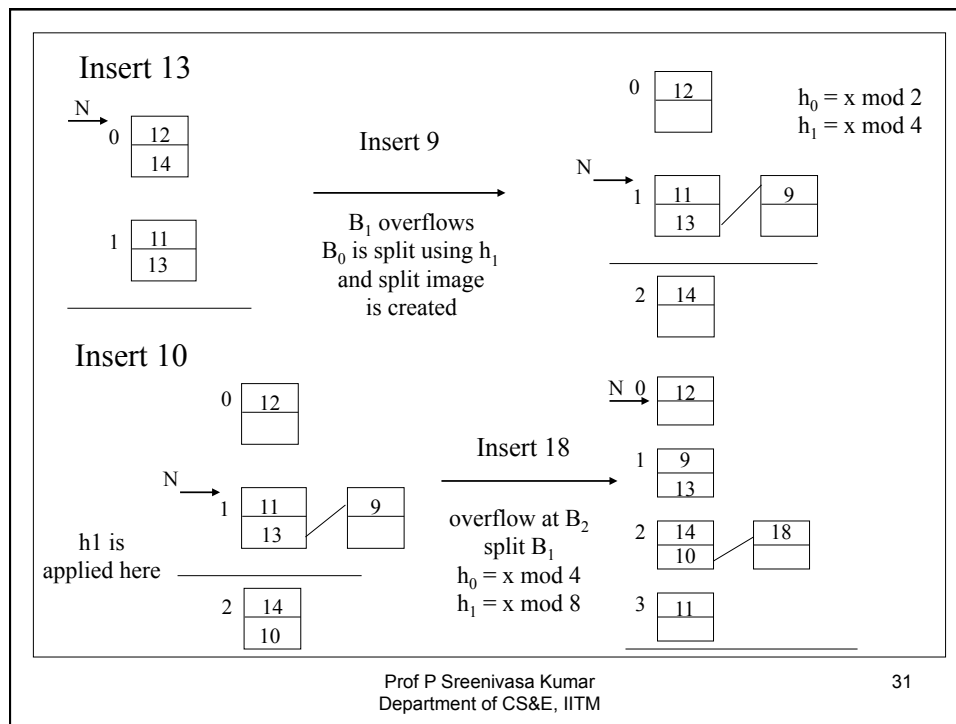


Insert 14



Prof P Sreenivasa Kumar
Department of CS&E, IITM

30



Index Structures

Index: A disk data structure

- enables efficient retrieval of a record given the value (s) of certain attributes
- indexing attributes

Primary Index:

Index built on *ordering key* field of a file

Clustering Index:

Index built on *ordering non-key* field of a file

Secondary Index:

Index built on any *non-ordering* field of a file

Primary Index

Can be built on ordered / sorted files

Index attribute – ordering key field (OKF)

Index Entry:

value of OKF for the <u>first record</u> of a block B_j	disk address of B_j
---	-----------------------

Index file: ordered file (sorted on OKF)

size: no. of blocks in the data file

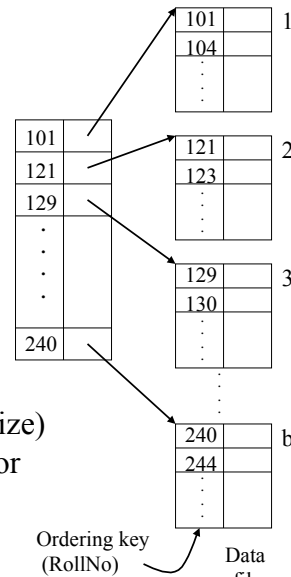
Index file blocking factor $BF_i = \lfloor B/(V+P) \rfloor$

(B-block size, V-OKF size, P-block pointer size)

- generally more than data file blocking factor

No of Index file blocks $b_i = \lceil b/BF_i \rceil$

(b - no. of data file blocks)



Prof P Sreenivasa Kumar
Department of CS&E, IITM

33

Record Access Using Primary Index

Given Ordering key field (OKF) value: x

Carry out binary search on the index file

m – value of OKF for the first record in the *middle block* k of the index file

$x < m$: do binary search on blocks $1, \dots, (k-1)$ of index file

$x \geq m$: if there are an index entries $(v_j, P_j), (v_{j+1}, P_{j+1})$ in block k such that $v_j \leq x < v_{j+1}$,

use the block pointer P_j , get the data file block and search for the data record with OKF value x

else

do binary search on blocks $k+1, \dots, b_i$ of index file

Maximum block accesses required: $\lceil \log_2 b_i \rceil$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

34

An Example

Data file:

No. of blocks $b = 9500$

Block size $B = 4KB$

OKF length $V = 15$ bytes

Block pointer length $p = 6$ bytes

Index file

No. of records $r_i = 9500$

Size of entry $V + P = 21$ bytes

Blocking factor $BF_i = \lfloor 4096/21 \rfloor = 195$

No. of blocks $b_i = \lceil r_i/BF_i \rceil = 49$

Max No. of block accesses for getting record using the primary index $\left| 1 + \lceil \log_2 b_i \rceil = 7 \right|$

Max No. of block accesses for getting record without using primary index $\left| \lceil \log_2 b \rceil = 14 \right|$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

35

Making the Index Multi-level

Index file – itself an ordered file

– another level of index can be built

Multilevel Index –

Successive levels of indices are built till the last level has one block

height – no. of levels

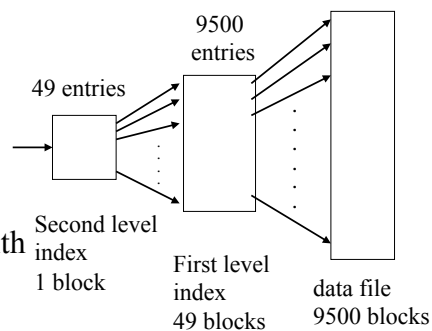
block accesses: height + 1

(no binary search required)

For the example data file:

No of block accesses required with multi-level primary index: 3

without any index: 14



Prof P Sreenivasa Kumar
Department of CS&E, IITM

36

Range Search, Insertion and Deletion

Range search on the ordering key field:

Get records with OKF value between x_1 and x_2 (inclusive)

Use the index to locate the record with OKF value x_1 and read succeeding records till OKF value exceeds x_2 .

Very efficient

Insertion: Data file – keep 25% of space in each block free

-- to take care of future insertions

index doesn't get changed

-- or use overflow chains for blocks that overflow

Deletion: Handle using deletion markers so that index doesn't get affected

Basically, avoid changes to index

Prof P Sreenivasa Kumar
Department of CS&E, IITM

37

Clustering Index

Built on ordered files where ordering field is *not a key*

Index attribute: ordering field (OF)

Index entry:

Distinct value V_i of the OF	address of the first block that has a record with OF value V_i
-----------------------------------	---

Index file: Ordered file (sorted on OF)

size – no. of distinct values of OF

Prof P Sreenivasa Kumar
Department of CS&E, IITM

38

Secondary Index

Built on any non-ordering field (NOF) of a data file.

Case I: NOF is also a key (Secondary key)

value of the NOF V_i	pointer to the record with V_i as the NOF value
------------------------	---

Case II: NOF is not a key: two options

(1)

value of the NOF V_i	pointer(s) to the record(s) with V_i as the NOF value
------------------------	---

(2)

value of the NOF V_i	pointer to a block that has pointer(s) to the record(s) with V_i as the NOF value
------------------------	---

Remarks:

(1) index entry – variable length record

(2) index entry – fixed length – One more level of indirection

Prof P Sreenivasa Kumar
Department of CS&E, IITM

39

Secondary Index (key)

Can be built on ordered and also other type of files

Index attribute: non-ordering key field

Index entry:

value of the NOF V_i	pointer to the <i>record</i> with V_i as the NOF value
------------------------	--

Index file: ordered file (sorted on NOF values)

No. of entries – same as the no. of *records* in the data file

Index file blocking factor $Bf_i = \left\lfloor \frac{B}{(V+P_r)} \right\rfloor$

(B: block size, V: length of the NOF,

P_r : length of a record pointer)

Index file blocks = $\lceil r/Bf_i \rceil$

(r – no. of records in the data file)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

40

An Example

Data file:

No. of records $r = 90,000$ Block size $B = 4\text{KB}$
Record length $R = 100$ bytes $BF = \lfloor 4096/100 \rfloor = 40$,
 $b = \lceil 90000/40 \rceil = 2250$
NOF length $V = 15$ bytes length of a record pointer $P_r = 7$ bytes

Index file :

No. of records $r_i = 90,000$ record length $= V + P_r = 22$ bytes
 $BF_i = \lfloor 4096/22 \rfloor = 186$ No. of blocks $b_i = \lceil 90000/186 \rceil = 484$

Max no. of block accesses to get a record
using the secondary index $1 + \lceil \log_2 b_i \rceil = 10$
Avg no. of block accesses to get a record
without using the secondary index $b/2 = 1125$

A very significant improvement

Prof P Sreenivasa Kumar
Department of CS&E, IITM

41

Multi-level Secondary Indexes

Secondary indexes can also be converted to multi-level indexes

First level index

– as many entries as there are records in the data file

First level index is an ordered file

so, in the second level index, the number of entries will be
equal to the number of *blocks* in the first level index
rather than the number of *records*

Similarly in other higher levels

Prof P Sreenivasa Kumar
Department of CS&E, IITM

42

Making the Secondary Index Multi-level

Multilevel Index –

Successive levels of indices are built
till the last level has one block

height – no. of levels

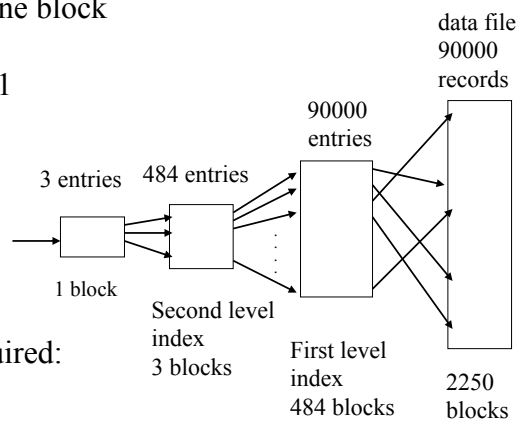
block accesses: height + 1

For the example data file:

No of block accesses required:

multi-level index: 4

single level index: 10



Prof P Sreenivasa Kumar
Department of CS&E, IITM

43

Index Sequential Access Method (ISAM) Files

ISAM files –

Ordered files with a multilevel primary/clustering index

Insertions:

Handled using overflow chains at data file blocks

Deletions:

Handled using deletion markers

Most suitable for files that are relatively static

If the files are dynamic, we need to go for dynamic multi-level index structures based on B⁺- trees

Prof P Sreenivasa Kumar
Department of CS&E, IITM

44

B⁺ - trees

Bayer & McCreight
Acta Informatica 1972

- Balanced search trees (self-balancing)
 - Internal nodes have variable number of children
 - All leaves are at the same level
 - Nodes – internal or leaf – are disk blocks
- Leaf node entries point to the actual data records
 - All leaf nodes are linked up as a list
- Internal node entries carry only index information
 - In B-trees, internal nodes carry data record pointers also
 - The fan-out in B-trees is less
- Make sure that blocks are always at least half filled
- Support both random and sequential access of records

Prof P Sreenivasa Kumar
Department of CS&E, IITM

45

Order

Order (m) of an Internal Node

- Order of an internal node is the maximum number of tree pointers held in it.
- Maximum of (m-1) keys can be present in an internal node

Order (m_{leaf}) of a Leaf Node

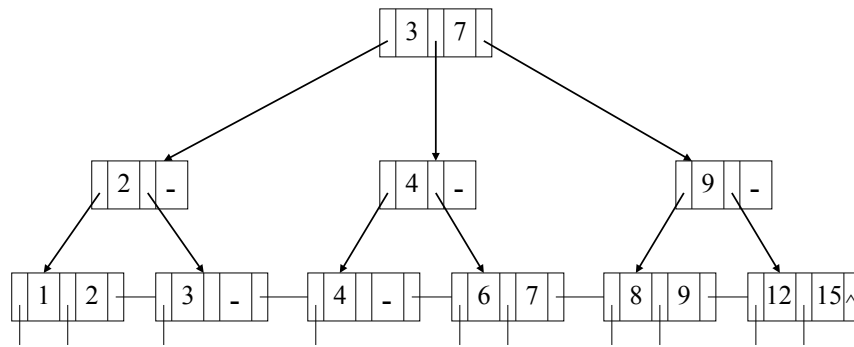
- Order of a leaf node is the maximum number of record pointers held in it. It is equal to the number of keys in a leaf node.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

46

Example B⁺- tree

$m = 3$ $m_{\text{leaf}} = 2$



Prof P Sreenivasa Kumar
Department of CS&E, IITM

47

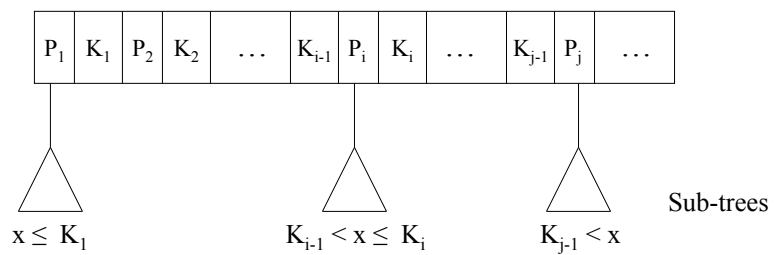
Internal Node Structure

$$\left\lceil \frac{m}{2} \right\rceil \leq j \leq m$$

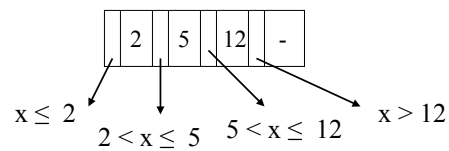
P_i : Tree pointer
(Block pointer)

K_i : Key value

m : Order(internal)



Example



Prof P Sreenivasa Kumar
Department of CS&E, IITM

48

Internal Nodes

An internal node of a B⁺- tree of order m :

- It contains at least $\lceil \frac{m}{2} \rceil$ pointers, except when it is the root node (Root node – a min of 2 pointers is ok)
- It contains at most m pointers.
- If it has P_1, P_2, \dots, P_j pointers with $K_1 < K_2 < K_3 \dots < K_{j-1}$ as keys, where $\lceil \frac{m}{2} \rceil \leq j \leq m$, then
 - P_1 points to the sub-tree with records having key value $x \leq K_1$
 - P_i ($1 < i < j$) points to the sub-tree with records having key value x such that $K_{i-1} < x \leq K_i$
 - P_j points to records with key value $x > K_{j-1}$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

49

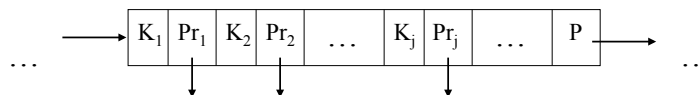
Leaf Node Structure

Structure of leaf node of B⁺- of order m_{leaf} :

- It contains one block pointer P to point to next leaf node
- At least $\lceil \frac{m_{\text{leaf}}}{2} \rceil$ record pointers and $\lceil \frac{m_{\text{leaf}}}{2} \rceil$ key values
- At most m_{leaf} record pointers and key values
- If a node has keys $K_1 < K_2 < \dots < K_j$ with $Pr_1, Pr_2 \dots Pr_j$ as record pointers and P as block pointer, then

Pr_i points to record with K_i as the search field value, $1 \leq i \leq j$

P points to next leaf block



Prof P Sreenivasa Kumar
Department of CS&E, IITM

50

Order Calculation

Block size: B, Size of Index field: V

Size of block pointer: P, Size of record pointer: P_r

Order of Internal node (m):

As there can be at most m block pointers and (m-1) keys

$$(m * P) + ((m-1) * V) \leq B$$

m can be calculated by using the above inequality (choose max)

Order of leaf node:

As there can be at most m_{leaf} record pointers and keys
with one block pointer in a leaf node,

m_{leaf} can be calculated by using the inequality: (choose max)

$$(m_{\text{leaf}} * (P_r + V)) + P \leq B$$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

51

Example Order Calculation

Given B = 512 bytes V = 8 bytes

P = 6 bytes P_r = 7 bytes. Then

Internal node order m = ?

$$m * P + ((m-1) * V) \leq B$$

$$m * 6 + ((m-1) * 8) \leq 512$$

$$14m \leq 520$$

$$m \leq 37$$

Leaf order m_{leaf} = ?

$$m_{\text{leaf}} (P_r + V) + P \leq 512$$

$$m_{\text{leaf}} (7 + 8) + 6 \leq 512$$

$$15m_{\text{leaf}} \leq 506$$

$$m_{\text{leaf}} \leq 33$$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

52

Insertion into B⁺- trees

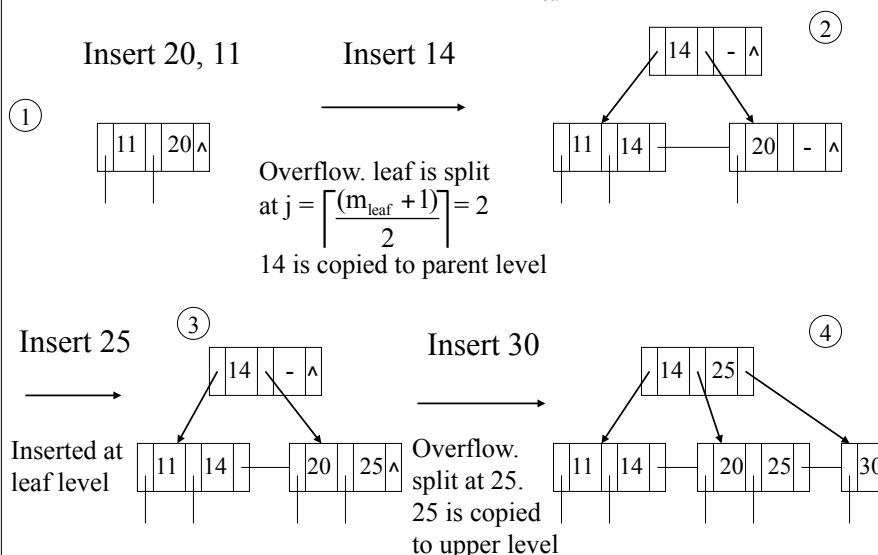
Every (key, record pointer) pair is inserted in an appropriate leaf
(Search for it)

- If a leaf node overflows:
 - Node is split at $j = \left\lceil \frac{(m_{\text{leaf}} + 1)}{2} \right\rceil$
 - First j entries are kept in original node
 - Entities from $j+1$ are moved to new node
 - j^{th} key value K_j is *replicated* in the parent of the leaf.
- If an internal node overflows:
 - Node is split at $j = \left\lceil \frac{(m + 1)}{2} \right\rceil$
 - Values and pointers up to P_j are kept in the original node
 - j^{th} key value K_j is *moved* to the parent of the internal node
 - P_{j+1} and the rest of entries are moved to a new node.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

53

Example of Insertions $m = 3$ $m_{\text{leaf}} = 2$

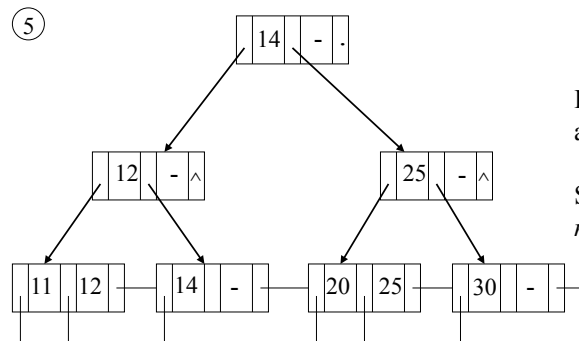


Prof P Sreenivasa Kumar
Department of CS&E, IITM

Insert 12

Overflow at leaf level.

- Split at leaf level,
- Triggers overflow at internal node
- Split occurs at internal node;



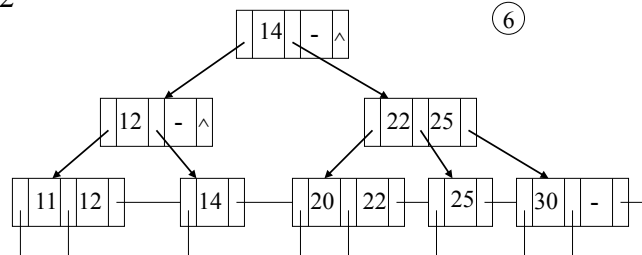
Internal node split
at $j = \left\lceil \frac{m}{2} \right\rceil$

Split at 14 and 14 is
moved up

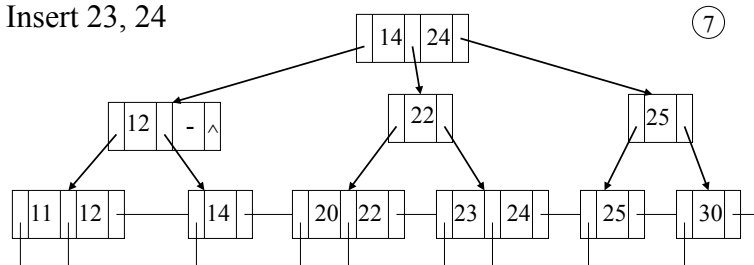
Prof P Sreenivasa Kumar
Department of CS&E, IITM

55

Insert 22



Insert 23, 24



Prof P Sreenivasa Kumar
Department of CS&E, IITM

56

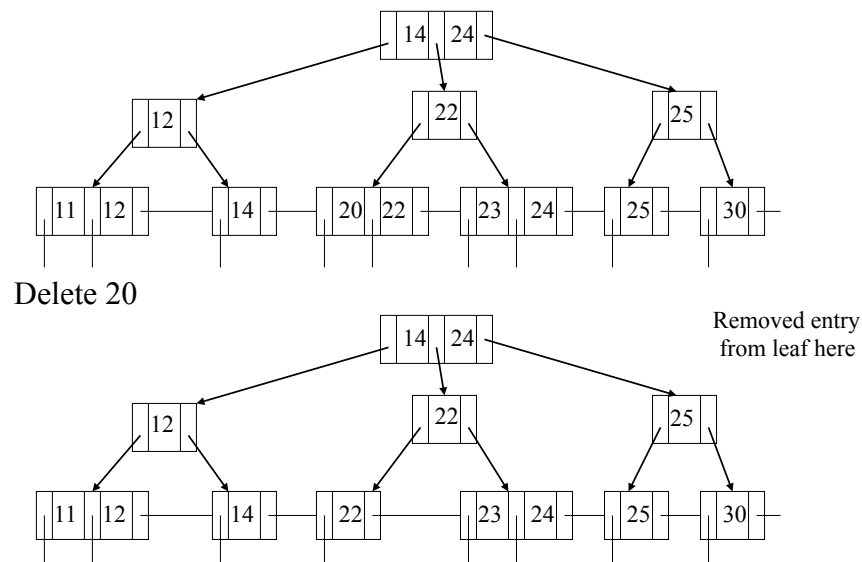
Deletion in B⁺ - trees

- Delete the entry from the leaf node
- Delete the entry if it is present in Internal node and replace with the entry to its right / right sibling.
- If underflow occurs after deletion
 - Distribute the entries from left sibling
 - if not possible – Distribute the entries from right sibling
 - if not possible – Merge the node with left and right sibling

Prof P Sreenivasa Kumar
Department of CS&E, IITM

57

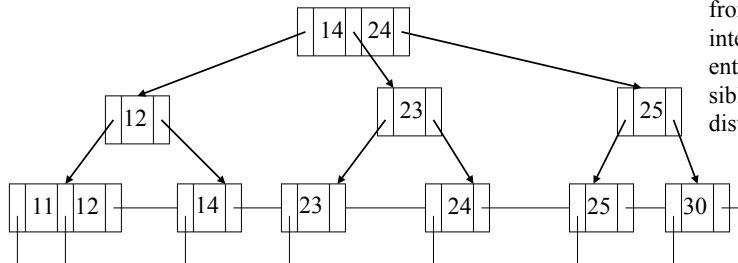
Example



Prof P Sreenivasa Kumar
Department of CS&E, IITM

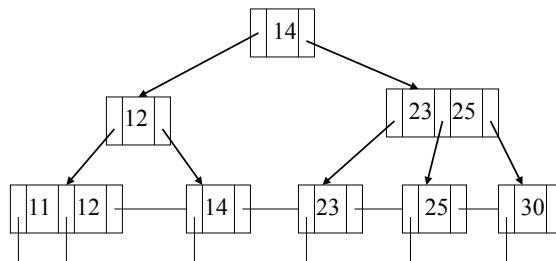
58

Delete 22



22 is removed from leaf and internal node entries from right sibling are distributed to left

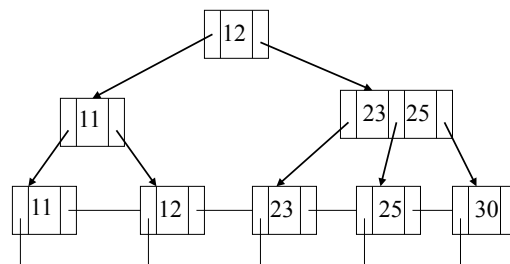
Delete 24



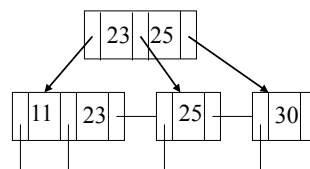
Prof P Sreenivasa Kumar
Department of CS&E, IITM

59

Delete 14



Delete 12



Level drop has occurred

Prof P Sreenivasa Kumar
Department of CS&E, IITM

60

Advantages of B⁺ - trees:

- 1) Any record can be fetched in equal number of disk accesses.
- 2) Range queries can be performed easily as leaves are linked up
- 3) Height of the tree is less as only keys are used for indexing
- 4) Supports both random and sequential access.

Disadvantages of B⁺ - trees:

Insert and delete operations are complicated

Root node becomes a *hotspot*

Prof P Sreenivasa Kumar
Department of CS&E, IITM

61

Parallel Access of Multiple Disks

Single Disk: high block access time: 6msec – 50msec

Why not use parallel access to improve performance?

RAID – Redundant Array of Independent Disks (current usage)

Redundant Array of Inexpensive Disks (early usage)

RAID techniques aim to improve performance and reliability

Two ideas are employed

- 1) Data Striping – distribute data on to multiple disks
Parallel reading of disks – faster data access
- 2) Add redundant data to help recover from disk crashes
Take help of error-recovery codes

Details follow ...

Prof P Sreenivasa Kumar
Department of CS&E, IITM

62

Data Striping

Data Striping – distribute data on multiple disks

Bit-level striping: i^{th} bit of each byte – stored on the i^{th} disk

Use 8 disks for 8 bits of a byte. // higher granularity is also possible

One (parallel) block read – 8 blocks of the data file

Transfer rate – eight times that of single disk

Read/write of a block – involves use of all the disks

Block-level striping: i^{th} block of data – i^{th} disk

Using n disks –

Single block access: n simultaneous block reads can happen

Multi-block access: n fold increase in transfer rate (parallel reads)

Downside: reliability of the set of disks comes down

Prof P Sreenivasa Kumar
Department of CS&E, IITM

63

Reliability of Multiple Disks

Reliability is modeled using Mean Time To Failure (MTTF)

An example scenario:

Mean Time To Failure (MTTF) of a disk: 2,40,000hrs

That is, probability of failure of a single disk in an hour: $1/2,40,000$

Probability of Failure of a single disk in a 100-disk set: $1/2,400$

MTTF of the 100-disk system is 2,400hrs = 100days ~ 3.3months!

This is unacceptable..

Prof P Sreenivasa Kumar
Department of CS&E, IITM

64

Mirroring disks to increase reliability

Mirroring – Each disk has a mirror disk – same data on both

If a disk fails – use the mirror of that disk till the original is replaced

One can improve reliability greatly:

- A disk with MTTF = 2,40,000hrs – mirrored with same kind of disk
- Probability of a disk failure in a particular hour: $2/2,40,000$
- Time to repair/copy a disk is, say, 24hrs
- Probability of disk failure while copying/repair: $24/2,40,000$
- Probability of a *data loss*: $(2/2,40,000) * (24/2,40,000) = 1/(12*10^8)$
- Or MTTF of the combination = $12*10^8$ hrs

Performance: reading: same as a single disk or better

Writing: same as single disk, both disks are updated in parallel

Prof P Sreenivasa Kumar
Department of CS&E, IITM

65

Reliability and performance with parity disks

Mirroring – High reliability; uses 50% more disks!

Get good reliability & also performance with fewer additional disks?

Idea: Store additional information to recover data of the failed disk

Error-correcting codes – parity bit (1 if #of 1's is odd, 0 otherwise)

Data: 1 0 1 1 0 0 1 0 - Parity Bit: 0 (#of 1's in Data & Parity is *even*)

Data: 1 0 0 1 1 0 1 1 - Parity Bit: 1 (#of 1's in Data & Parity is *even*)

Parity block: (Assuming block-level data striping with N disks)

The i^{th} bit of the parity block j : parity of the i^{th} bits of block j on all disks

Parity Disk – has parity blocks for all data blocks

If a disk k fails: Set the i^{th} bit of block j using i^{th} parity bit of block j

Do this for all blocks to recover data of disk k !

N data disks, one extra disk – good performance and reliability!

Prof P Sreenivasa Kumar
Department of CS&E, IITM

66

Distributed Parity

N data disks and 1 redundant (parity) disk

- Very good performance and protection against single-disk crash
- Updating *any* data block – requires updating the parity disk
- Usage of parity disk – high and it ages faster!

Can we distribute the parity information?

Use each disk as a redundant (parity) disk for some *part* of the data!

Say, we have $D_0, D_1, D_2, \dots, D_5$ – 6 disks with, say, 60 cylinders each

Use each as the redundant disk for 1/6 of data:

Cyl# 0, 6, 12, ... of D_0 – parity blocks for other disk cyl# 0, 6, 12, ...

Cyl# 1, 7, 13, ... of D_1 – parity blocks for other disk cyl# 1, 7, 13, ...

Etc...

This is called *distributed parity* – disk usage is uniform!

Prof P Sreenivasa Kumar
Department of CS&E, IITM

67

Standard RAID Levels

RAID-0 – Bit-level striping; No parity data; No mirroring

RAID-1 – Mirrored disks; No parity; No data striping

RAID-2 – Bit-level striping; Redundancy using Hamming codes

Not in much use currently.

RAID-3 – Byte-level striping; dedicated parity disk

Not in common use currently.

RAID-4 – Block-level striping; dedicated parity disk

RAID-5 – Block-level striping; distributed parity

RAID-6 – Block-level striping; double distributed parity;

Up to 2 disk crashes can be tolerated

Prof P Sreenivasa Kumar
Department of CS&E, IITM

68

Storage Area Networks (SAN)

Specialized computing systems for providing large-scale storage

- Dedicated hardware and software
- Shared across several servers
- Connected to servers through a dedicated high-speed network using special optical cables – Fiber channels
- Block-level data storage
- Internally use a large number of disks under a suitable RAID
- Offer SCSI (Small Computer System Interface) interface to servers
- Details are beyond the scope of this course

Database Design and Normal Forms

Database Design

- coming up with a “good” DB scheme is very important

How do we characterize the “goodness” of a schema ?

If two or more alternative schemas are available

how do we compare them ?

What are the problems with “bad” schema designs ?

Normal Forms:

Each normal form specifies certain conditions

If the conditions are satisfied by the schema

certain kind of problems are avoided

Details follow....

Prof P Sreenivasa Kumar
Department of CS&E, IITM

1

An Example

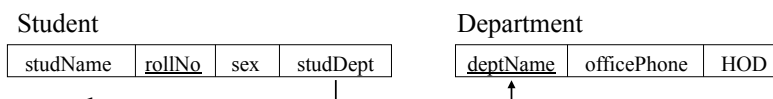
student relation with attributes: studName, rollNo, sex, studDept

department relation with attributes: deptName, officePhone, hod

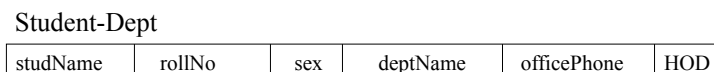
Several students belong to a department.

studDept gives the name of the student's department.

Correct schema:



Incorrect schema:



What are the problems that arise ?

Prof P Sreenivasa Kumar
Department of CS&E, IITM

2

Problems with bad schema

Student-Dept(studName, rollNo, sex, studDept, deptName, officePhone, hod)

Redundant storage of data:

Office Phone & HOD info - stored redundantly

- once with each student that belongs to the department
- wastage of disk space

A program that updates Office Phone of a department

- must change it at several places
 - more running time
 - error – prone

Transactions running on a database

- must take as short time as possible to increase transaction throughput

Prof P Sreenivasa Kumar
Department of CS&E, IITM

3

Update Anomalies

Another kind of problems with bad schema

Insertion anomaly:

No way of inserting info about a new department unless we also enter details of a (dummy) student in the department

Deletion anomaly:

If all students of a certain department leave and we delete their tuples, information about the department itself is lost

Update Anomaly:

Updating officePhone of a department

- value in several tuples needs to be changed
- if a tuple is missed - inconsistency in data

Prof P Sreenivasa Kumar
Department of CS&E, IITM

4

Normal Forms

First Normal Form (1NF) - included in the definition of a relation

Second Normal Form (2NF)

Third Normal Form (3NF)

Boyce-Codd Normal Form (BCNF)

} defined in terms of
functional dependencies

Fourth Normal Form (4NF) - defined using multivalued
dependencies

Fifth Normal Form (5NF) or Project Join Normal Form (PJNF)
defined using join dependencies

Prof P Sreenivasa Kumar
Department of CS&E, IITM

5

Functional Dependencies

A functional dependency (FD) $X \rightarrow Y$ [where $(X \subseteq R, Y \subseteq R)$]
(read as X *determines* Y)

is said to hold on a schema R if

in *any* instance r on R ,

if two tuples t_1, t_2 ($t_1 \neq t_2, t_1 \in r, t_2 \in r$)

agree on X i.e. $t_1[X] = t_2[X]$

then they also agree on Y i.e. $t_1[Y] = t_2[Y]$

$t_1[X]$ – the sub-tuple of t_1 consisting of values of attributes in X

Note: If $K \subset R$ is a key for R then for any $A \in R$,

$K \rightarrow A$

holds because the above ifthen condition is
vacuously true

Prof P Sreenivasa Kumar
Department of CS&E, IITM

6

Functional Dependencies – Examples

Consider the schema:

Student(studName, rollNo, sex, dept, hostelName, roomNo)

Since rollNo is a key, $\text{rollNo} \rightarrow \{\text{studName}, \text{sex}, \text{dept}, \text{hostelName}, \text{roomNo}\}$

Suppose that each student is given a hostel room exclusively, then
 $\text{hostelName}, \text{roomNo} \rightarrow \text{rollNo}$

Suppose boys and girls are accommodated in separate hostels, then
 $\text{hostelName} \rightarrow \text{sex}$

Does $\text{Sex} \rightarrow \text{hostelName}$?

FDs are additional constraints that can be specified by designers

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

7

Trivial / Non-Trivial FDs and Notation

An FD $X \rightarrow Y$ where $Y \subseteq X$

- called a *trivial* FD, as it always holds good

An FD $X \rightarrow Y$ where $Y \not\subseteq X$

- *non-trivial* FD

An FD $X \rightarrow Y$ where $X \cap Y = \Phi$

- *completely non-trivial* FD

Notational Convention:

(Low-end alphabets) A, B, C, D, ... and their subscripted versions
 -- denote individual attributes

(High-end alphabets) Z, Y, X, W, ... and their subscripted versions
 --- denote sets of attributes

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

8

FDs – Examples

Consider the scheme `preRequisite(preReqCourse, courseId)`

Does `preReqCourse → courseId` ?

No, as a course might be pre-requisite for many courses

Does `courseId → preReqCourse` ?

No, a course may have many pre-requisite courses

So, it is possible that no FDs hold on some schema

Prof P Sreenivasa Kumar
Department of CS&E, IITM

9

FDs – Examples

Consider the scheme:

`Student-dept(rollNo, name, sex, deptName, officePhone, Hod)`

The key is `rollNo`, so

`rollNo → name, sex, deptName, officePhone, Hod`

Any more FDs hold?

`deptName → officePhone, Hod`

`Hod → deptName, officePhone`

(Assuming that each professor heads at most one department)

`officePhone → deptName, Hod`

No other FDs hold

Prof P Sreenivasa Kumar
Department of CS&E, IITM

10

Deriving new FDs

Given that a set of FDs F holds on R
we can infer that a certain new FD must also hold on R

For instance,
given that $X \rightarrow Y, Y \rightarrow Z$ hold on R
we can infer that $X \rightarrow Z$ must also hold

How to systematically obtain all such new FDs ?

Unless *all* FDs are known, a relation schema is not fully specified

Prof P Sreenivasa Kumar
Department of CS&E, IITM

11

Entailment Relation

We say that a set of FDs $F \models \{X \rightarrow Y\}$
(read as F *entails* $X \rightarrow Y$ or
 F *logically implies* $X \rightarrow Y$)
if in every instance r of R on which FDs F hold,
FD $X \rightarrow Y$ also holds.

Researcher W W Armstrong came up with several inference rules
for deriving new FDs from a given set of FDs

We define $F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$
 F^+ : Closure of F

William Ward Armstrong: Dependency Structures of Data Base
Relationships, page 580–583. IFIP Congress, 1974.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

12

Armstrong's Inference Rules (1/2) (aka Armstrong's Axioms)

1. Reflexive rule

$F \models \{X \rightarrow Y \mid Y \subseteq X\}$ for any X . Trivial FDs.

2. Augmentation rule

$\{X \rightarrow Y\} \models \{XZ \rightarrow YZ\}, Z \subseteq R$. Here, XZ denotes $X \cup Z$

3. Transitive rule

$\{X \rightarrow Y, Y \rightarrow Z\} \models \{X \rightarrow Z\}$

4. Decomposition or Projective rule

$\{X \rightarrow YZ\} \models \{X \rightarrow Y\}$ // RHS can be $\{X \rightarrow Z\}$ also.

5. Union or Additive rule

$\{X \rightarrow Y, X \rightarrow Z\} \models \{X \rightarrow YZ\}$

6. Pseudo transitive rule

$\{X \rightarrow Y, WY \rightarrow Z\} \models \{WX \rightarrow Z\}$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

13

Armstrong's Inference Rules (2/2)

Rules 4, 5, 6 are not really necessary.

For instance, Rule 5: $\{X \rightarrow Y, X \rightarrow Z\} \models \{X \rightarrow YZ\}$ can be
proved using 1, 2, 3 alone

- 1) $X \rightarrow Y$
- 2) $X \rightarrow Z$
- } given
- 3) $X \rightarrow XY$ Augmentation rule on 1
- 4) $XY \rightarrow ZY$ Augmentation rule on 2
- 5) $X \rightarrow ZY$ Transitive rule on 3, 4.

Similarly, 4, 6 can be shown to be unnecessary.

But it is useful to have 4, 5, 6 as short-cut rules

Prof P Sreenivasa Kumar
Department of CS&E, IITM

14

Sound and Complete Inference Rules

Armstrong showed that

Rules (1), (2) and (3) are sound and complete.

These are called Armstrong's Axioms (AA)

$F_{AA} = \{ X \rightarrow Y \mid X \rightarrow Y \text{ can be derived from } F \text{ using AA} \}$

Soundness: ($F_{AA} \subseteq F^+$)

Every new FD $X \rightarrow Y$ derived from a given set of FDs F using Armstrong's Axioms is such that $F \models \{X \rightarrow Y\}$

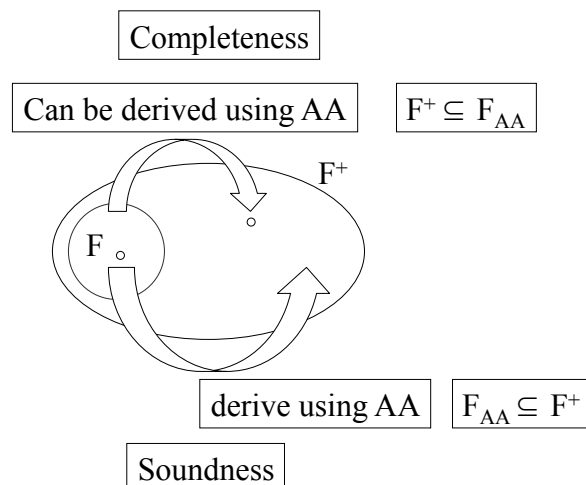
Completeness: ($F^+ \subseteq F_{AA}$)

Any FD $X \rightarrow Y$ logically implied by F (i.e. $F \models \{X \rightarrow Y\}$) can be derived from F using Armstrong's Axioms

Prof P Sreenivasa Kumar
Department of CS&E, IITM

15

Soundness and Completeness of AA



Prof P Sreenivasa Kumar
Department of CS&E, IITM

16

Proving Soundness

Suppose $X \rightarrow Y$ is derived from F using AA in some n steps.
If each step is correct then overall deduction would be correct.

Single step: Apply Rule (1) or (2) or (3)

Rule (1) – Reflexive Rule. Obviously results in correct FDs

Rule (2) – $\{X \rightarrow Y\} \models \{XZ \rightarrow YZ\}, Z \subseteq R$

Suppose $t_1, t_2 \in r$ agree on XZ

$\Rightarrow t_1, t_2$ agree on X

$\Rightarrow t_1, t_2$ agree on Y (since $X \rightarrow Y$ holds on r)

$\Rightarrow t_1, t_2$ agree as YZ

Hence Rule (2) gives rise to correct FDs

Rule (3) – $\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

Suppose $t_1, t_2 \in r$ agree on X

$\Rightarrow t_1, t_2$ agree on Y (since $X \rightarrow Y$ holds)

$\Rightarrow t_1, t_2$ agree on Z (since $Y \rightarrow Z$ holds)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

17

Proving Completeness of Armstrong's Axioms (1/4)

Define X_F^+ (closure of X wrt F)

$= \{A \mid X \rightarrow A \text{ can be derived from } F \text{ using AA}\}, A \in R$

X_F^+ is the set of all attributes that occur on

the rhs for an FD whose lhs is X , as per AA (wrt F)

Claim1:

$X \rightarrow Y$ can be derived from F using AA iff $Y \subseteq X^+$

(If) Let $Y = \{A_1, A_2, \dots, A_n\}$. $Y \subseteq X^+$

$\Rightarrow X \rightarrow A_i$ can be derived from F using AA ($1 \leq i \leq n$)

By union rule, it follows that $X \rightarrow Y$ can be derived from F .

(Only If) $X \rightarrow Y$ can be derived from F using AA

By projective rule $X \rightarrow A_i$ ($1 \leq i \leq n$)

Thus by definition of X^+ , $A_i \in X^+$

$\Rightarrow Y \subseteq X^+$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

18

Completeness of Armstrong's Axioms (2/4)

Completeness:

$(F \models \{X \rightarrow Y\}) \Rightarrow X \rightarrow Y$ follows from F using AA

We will prove the contrapositive:

$X \rightarrow Y$ can't be derived from F using AA

$\Rightarrow F \not\models \{X \rightarrow Y\}$

$\Rightarrow \exists$ a relation instance r on R st all the FDs of F hold on r but $X \rightarrow Y$ doesn't hold.

Consider the relation instance r with just two tuples:

	X^+ attributes				Other attributes			
r:	1	1	1	...1	1	1	1	...1
	1	1	1	...1	0	0	0	...0

Prof P Sreenivasa Kumar
Department of CS&E, IITM

19

Completeness Proof (3/4)

Claim 2: All FDs of F are satisfied by r

Suppose not. Let $W \rightarrow Z$ in F be an FD not satisfied by r

Then $W \subseteq X^+$ and $Z \not\subseteq X^+$

Let $A \in Z - X^+$

Now, $X \rightarrow W$ follows from F using AA as $W \subseteq X^+$ (claim 1)

$X \rightarrow Z$ follows from F using AA by transitive rule

$Z \rightarrow A$ follows from F using AA by reflexive rule as $A \in Z$

$X \rightarrow A$ follows from F using AA by transitive rule

By definition of closures, A must belong to X^+

- a contradiction.

Hence the claim.

r:	1	1	1	...1	1	1	1	...1
	1	1	1	...1	0	0	0	...0
	X^+				$R - X^+$			

Prof P Sreenivasa Kumar
Department of CS&E, IITM

20

Completeness Proof (4/4)

Claim 3: $X \rightarrow Y$ is not satisfied by r

Suppose not

Because of the structure of r , $Y \subseteq X^+$

$\Rightarrow X \rightarrow Y$ can be derived from F using AA

contradicting the assumption about $X \rightarrow Y$

Hence the claim

Thus, whenever $X \rightarrow Y$ doesn't follow from F using AA,

F doesn't logically imply $X \rightarrow Y$

Armstrong's Axioms are complete.

$$r: \begin{array}{cccccccc} 1 & 1 & 1 & \dots & 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 & 0 & 0 & 0 & \dots & 0 \end{array}$$

$\underbrace{\hspace{10em}}_{X^+} \quad \underbrace{\hspace{10em}}_{R - X^+}$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

21

Consequence of Completeness of AA

Attribute Closure wrt F – for a given set of attributes X :

$$X_F^+ = \{A \mid X \rightarrow A \text{ follows from } F \text{ using AA}\}$$

$$= \{A \mid F \models X \rightarrow A\}$$

Similarly

$$F^+ = \{X \rightarrow Y \mid F \models X \rightarrow Y\}$$

$$= \{X \rightarrow Y \mid X \rightarrow Y \text{ follows from } F \text{ using AA}\}$$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

22

Computing closures

The size of F^+ can sometimes be exponential in the size of F .

For instance, $F = \{A \rightarrow B_1, A \rightarrow B_2, \dots, A \rightarrow B_n\}$

$F^+ = \{A \rightarrow X\}$ where $X \subseteq \{B_1, B_2, \dots, B_n\}$.

Thus $|F^+| = 2^n$

Computing F^+ : computationally expensive

Fortunately, checking if $X \rightarrow Y \in F^+$

can be done by checking if $Y \subseteq X_F^+$

Computing attribute closure (X_F^+) is computationally easier

Prof P Sreenivasa Kumar
Department of CS&E, IITM

23

Computing X_F^+

We compute a sequence of sets X_0, X_1, \dots as follows:

$X_0 = X$; // X is the given set of attributes

$X_{i+1} = X_i \cup \{A \mid \text{there is a FD } Y \rightarrow Z \text{ in } F$
such that $Y \subseteq X_i \text{ and } A \in Z\}$

To get new attributes into X_{i+1} , we use Transitive Rule and
we can only use that!

Since $X_0 \subseteq X_1 \subseteq X_2 \subseteq \dots \subseteq X_i \subseteq X_{i+1} \subseteq \dots \subseteq R$, and R is finite,
There is an integer i such that $X_i = X_{i+1} = X_{i+2} = \dots$

X_F^+ is equal to such X_i .

Computing X_F^+ can be done in polynomial time.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

24

Attribute Closures – An Example

Consider a scheme R and the FDs: (Data redundancy exists in R)

$R = (\text{rollNo}, \text{name}, \text{advisorId}, \text{advisorName}, \text{courseId}, \text{grade})$

FDs = { $\text{rollNo} \rightarrow \text{name}$; $\text{rollNo} \rightarrow \text{advisorId}$;
 $\text{advisorId} \rightarrow \text{advisorName}$;
 $\text{rollNo}, \text{courseId} \rightarrow \text{grade}$ }

$\{\text{rollNo}\}^+ = \{\text{rollNo}, \text{name}, \text{advisorId}, \text{advisorName}\}$

$\{\text{rollNo}, \text{courseId}\}^+ = \{\text{rollNo}, \text{name}, \text{advisorId}, \text{advisorName}, \text{courseId}, \text{grade}\} = R$

So $\{\text{rollNo}, \text{courseId}\}$ is the key for R.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

25

Normal Forms – 2NF

Full functional dependency:

An FD $X \rightarrow A$ for which there is no proper subset Y of X
 such that $Y \rightarrow A$
 (A is said to be *fully functionally* dependent on X)

2NF: A relation schema R is in 2NF if
 every *non-prime* attribute is fully functionally dependent
 on any key of R

Prime attribute: A attribute that is part of some key

Non-prime attribute: An attribute that is not part of any key

Prof P Sreenivasa Kumar
Department of CS&E, IITM

26

Example 1: 2NF

student(rollNo, name, dept, sex, hostelName, roomNo, admitYear)

Assumptions:

Each student is allotted a single-occupancy room.

A room is identified by values of attributes hostelName, roomNo.

Boys and girls are accommodated in separate hostels.

Keys: rollNo, (hostelName, roomNo)

Not in 2NF as hostelName → sex

Decompose:

student(rollNo, name, dept, hostelName, roomNo, admitYear)

hostelDetail(hostelName, sex)

- These are both in 2NF

Prof P Sreenivasa Kumar
Department of CS&E, IITM

27

Example 2: 2NF

book(authorName, title, authorAffiliation, ISBN, publisher, pubYear)

Assumptions: A book has exactly one author.

Author can be uniquely identified by value of attribute authorName

AuthorAffiliation is the organization to which the author is *currently* associated with.

An author is associated with *exactly one* organization at any time.

Keys: (authorName, title), ISBN

Not in 2NF as authorName → authorAffiliation

(authorAffiliation is not fully functionally dependent on the first key)

Decompose:

book(authorName, title, ISBN, publisher, pubYear)

authorInfo(authorName, authorAffiliation) -- both in 2NF

Prof P Sreenivasa Kumar
Department of CS&E, IITM

28

Transitive Dependencies

Transitive dependency:

An FD $X \rightarrow Y$ in a relation schema R for which there is a set of attributes $Z \subseteq R$ such that

$X \rightarrow Z$ and $Z \rightarrow Y$ and Z is not a subset of any key of R

studentDept(rollNo, name, dept, hostelName, roomNo, headDept)

Keys: rollNo, (hostelName, roomNo)

rollNo \rightarrow dept; dept \rightarrow headDept hold

So, rollNo \rightarrow headDept is a transitive dependency

Head of the dept of dept D is stored redundantly in every tuple where D appears.

Relation is in 2NF but redundancy still exists.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

29

Normal Forms – 3NF

Relation schema R is in 3NF if it is in 2NF and no non-prime attribute of R is transitively dependent on any key of R

studentDept(rollNo, name, dept, hostelname, roomNo, headDept)
is not in 3NF

Decompose: student(rollNo, name, dept, hostelName, roomNo)
deptInfo(dept, headDept)

both in 3NF

Redundancy in data storage - removed

Prof P Sreenivasa Kumar
Department of CS&E, IITM

30

Another definition of 3NF

Relation schema R is in 3NF if for any nontrivial FD $X \rightarrow A$ either (i) X is a superkey or (ii) A is prime.

Suppose some R violates the above definition

\Rightarrow There is an FD $X \rightarrow A$ for which both (i) and (ii) are false

\Rightarrow X is not a superkey and A is non-prime attribute

Two cases (mutually exclusive) arise:

- 1) X is contained in a key – A is not fully functionally dependent on this key

- violation of 2NF condition and hence can not be in 3NF

- 2) X is not contained in a key

$K \rightarrow X, X \rightarrow A$ is a case of transitive dependency

(K – any key of R) ; hence can not be in 3NF

Prof P Sreenivasa Kumar
Department of CS&E, IITM

31

Motivating example for BCNF

gradeInfo (rollNo, studName, course, grade)

Suppose the following FDs hold:

- 1) rollNo, course \rightarrow grade

Keys:

- 2) studName, course \rightarrow grade

(rollNo, course)

- 3) rollNo \rightarrow studName

(studName, course)

- 4) studName \rightarrow rollNo

(Assumption: No two students have the same name)

For 1, 2 lhs is a key. For 3, 4 rhs is prime; so gradeInfo is in 3NF

But studName is stored redundantly along with every course being done by the student.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

32

Boyce - Codd Normal Form (BCNF)

Relation schema R is in BCNF if for every nontrivial FD $X \rightarrow A$, X is a superkey of R.

In gradeInfo, FDs 3, 4 are nontrivial but lhs is not a superkey
So, gradeInfo is not in BCNF

Decompose:

gradeInfo (rollNo, course, grade)

studInfo (rollNo, studName)

Redundancy allowed by 3NF is disallowed by BCNF

BCNF is stricter than 3NF

3NF is stricter than 2NF

Prof P Sreenivasa Kumar
Department of CS&E, IITM

33

Decomposition of a relation schema

If R doesn't satisfy a particular normal form,
we decompose R into smaller schemas

What's a decomposition?

$R = (A_1, A_2, \dots, A_n)$

$D = (R_1, R_2, \dots, R_k)$ st $R_i \subseteq R$ and $R = R_1 \cup R_2 \cup \dots \cup R_k$
(R_i 's need not be disjoint)

Replacing R by R_1, R_2, \dots, R_k is the process of decomposing R

Ex: gradeInfo (rollNo, studName, course, grade)

R_1 : gradeInfo (rollNo, course, grade)

R_2 : studInfo (rollNo, studName)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

34

Desirable Properties of Decompositions

Not all decomposition of a relational scheme R are useful

We require two properties to be satisfied

(i) Lossless join property

- the information in an instance r of R must be preserved in the instances r_1, r_2, \dots, r_k where $r_i = \Pi_{R_i}(r)$

(ii) Dependency preserving property

- if a set F of dependencies hold on R it should be possible to enforce F on an instance r by enforcing appropriate dependencies on each r_i

Prof P Sreenivasa Kumar
Department of CS&E, IITM

35

Lossless join property

F – set of FDs that hold on R

R – decomposed into R_1, R_2, \dots, R_k

Decomposition is lossless wrt F if

for every relation instance r on R satisfying F,

$$r = \Pi_{R_1}(r) * \Pi_{R_2}(r) * \dots * \Pi_{R_k}(r)$$

$R = (A, B, C); R_1 = (A, B); R_2 = (B, C)$

Lossless joins
are also called
non-additive joins

Original info
is distorted

r:	A	B	C	r ₁ :	A	B	r ₂ :	B	C	r ₁ *r ₂ :	A	B	C
	a ₁	b ₁	c ₁		a ₁	b ₁		b ₁	c ₁		a ₁	b ₁	c ₁
	a ₂	b ₂	c ₂		a ₂	b ₂		b ₂	c ₂		a ₁	b ₁	c ₃
	a ₃	b ₁	c ₃		a ₃	b ₁		b ₁	c ₃		a ₂	b ₂	c ₂
											a ₃	b ₁	c ₁
											a ₃	b ₁	c ₃

Lossy join

Spurious tuples

Prof P Sreenivasa Kumar
Department of CS&E, IITM

36

Dependency Preserving Decompositions

Decomposition $D = (R_1, R_2, \dots, R_k)$ of schema R *preserves* a set of dependencies F if

$$(\Pi_{R_1}(F) \cup \Pi_{R_2}(F) \cup \dots \cup \Pi_{R_k}(F))^+ = F^+$$

Here, $\Pi_{R_i}(F) = \{ (X \rightarrow Y) \in F^+ \mid X \subseteq R_i, Y \subseteq R_i \}$
(It is called the projection of F onto R_i)

Informally, any FD that logically follows from F must also logically follow from the union of projections of F onto R_i 's. Then, D is called dependency preserving.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

37

An example

Schema $R = (A, B, C)$

FDs $F = \{A \rightarrow B, B \rightarrow C, C \rightarrow A\}$

Decomposition $D = (R_1 = \{A, B\}, R_2 = \{B, C\})$

$\Pi_{R_1}(F) = \{A \rightarrow B, B \rightarrow A\}$

$\Pi_{R_2}(F) = \{B \rightarrow C, C \rightarrow B\}$

$$(\Pi_{R_1}(F) \cup \Pi_{R_2}(F))^+ = \{A \rightarrow B, B \rightarrow A, \\ B \rightarrow C, C \rightarrow B, \\ A \rightarrow C, C \rightarrow A\} = F^+$$

Hence Dependency preserving

Prof P Sreenivasa Kumar
Department of CS&E, IITM

38

Testing for lossless decomposition property(1/6)

R – given schema with attributes A_1, A_2, \dots, A_n

F – given set of FDs

D – $\{R_1, R_2, \dots, R_m\}$ given decomposition of R

Is D a lossless decomposition?

Create an $m \times n$ matrix S with columns labeled as A_1, A_2, \dots, A_n
and rows labeled as R_1, R_2, \dots, R_m

Initialize the matrix as follows:

set $S(i,j)$ as symbol b_{ij} for all i,j .

if A_j is in the scheme R_i , then set $S(i,j)$ as symbol a_j , for all i,j

Prof P Sreenivasa Kumar
Department of CS&E, IITM

39

Testing for lossless decomposition property(2/6)

After S is initialized, we carry out the following process on it:

repeat

for each functional dependency $U \rightarrow V$ in F **do**

for all rows in S which agree on U -attributes **do**

make the symbols in each V - attribute column

the *same* in all the rows as follows:

if any of the rows has an “ a ” symbol for the column

set the other rows to the same “ a ” symbol in the column

else // if no “ a ” symbol exists in any of the rows

choose one of the “ b ” symbols that appears

in one of the rows for the V -attribute and

set the other rows to that “ b ” symbol in the column

until no changes to S

At the end, if there exists a row with all “ a ” symbols then D is lossless otherwise D is a lossy decomposition

Prof P Sreenivasa Kumar
Department of CS&E, IITM

40

Testing for lossless decomposition property(3/6)

$R = (\text{rollNo}, \text{name}, \text{advisor}, \text{advisorName}, \text{course}, \text{grade})$

$FD's = \{ \text{rollNo} \rightarrow \text{name}; \text{rollNo} \rightarrow \text{advisor}; \text{advisor} \rightarrow \text{advisorName}$
 $\text{rollNo}, \text{course} \rightarrow \text{grade} \}$

$D : \{ R_1 = (\text{rollNo}, \text{name}, \text{advisor}), R_2 = (\text{advisor}, \text{advisorName}),$
 $R_3 = (\text{rollNo}, \text{course}, \text{grade}) \}$

Matrix S : (Initial values)

	rollNo	name	advisor	advisor Name	course	grade
R_1	a_1	a_2	a_3	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	b_{32}	b_{33}	b_{34}	a_5	a_6

Prof P Sreenivasa Kumar
Department of CS&E, IITM

41

Testing for lossless decomposition property(4/6)

$R = (\text{rollNo}, \text{name}, \text{advisor}, \text{advisorDept}, \text{course}, \text{grade})$

$FD's = \{ \text{rollNo} \rightarrow \text{name}; \text{rollNo} \rightarrow \text{advisor}; \text{advisor} \rightarrow \text{advisorName}$
 $\text{rollNo}, \text{course} \rightarrow \text{grade} \}$

$D : \{ R_1 = (\text{rollNo}, \text{name}, \text{advisor}), R_2 = (\text{advisor}, \text{advisorName}),$
 $R_3 = (\text{rollNo}, \text{course}, \text{grade}) \}$

Matrix S : (After enforcing $\text{rollNo} \rightarrow \text{name}$ & $\text{rollNo} \rightarrow \text{advisor}$)

	rollNo	name	advisor	advisor Name	course	grade
R_1	a_1	a_2	a_3	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	b_{32}	b_{33}	b_{34}	a_5	a_6

Prof P Sreenivasa Kumar
Department of CS&E, IITM

42

Testing for lossless decomposition property(5/6)

$R = (\text{rollNo}, \text{name}, \text{advisor}, \text{advisorDept}, \text{course}, \text{grade})$

$FD's = \{\text{rollNo} \rightarrow \text{name}; \text{rollNo} \rightarrow \text{advisor}; \text{advisor} \rightarrow \text{advisorName}$
 $\text{rollNo}, \text{course} \rightarrow \text{grade}\}$

$D : \{ R_1 = (\text{rollNo}, \text{name}, \text{advisor}), R_2 = (\text{advisor}, \text{advisorName}),$
 $R_3 = (\text{rollNo}, \text{course}, \text{grade}) \}$

Matrix S : (After enforcing $\text{advisor} \rightarrow \text{advisorName}$)

	rollNo	name	advisor	advisor Name	course	grade
R_1	a_1	a_2	a_3	b_{14} a_4	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	b_{25}	b_{26}
R_3	a_1	b_{32} a_2	b_{33} a_3	b_{34} a_4	a_5	a_6

No more changes. Third row with all a symbols. So a lossless join.

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

43

Testing for lossless decomposition property(6/6)

R – given schema. F – given set of FDs

The decomposition of R into R_1, R_2 is lossless wrt F if and only if
 either $R_1 \cap R_2 \rightarrow (R_1 - R_2)$ belongs to F^+ or
 $R_1 \cap R_2 \rightarrow (R_2 - R_1)$ belongs to F^+

Example:

$\text{gradeInfo}(\text{rollNo}, \text{studName}, \text{course}, \text{grade})$

with FDs = $\{\text{rollNo}, \text{course} \rightarrow \text{grade}; \text{studName}, \text{course} \rightarrow \text{grade};$
 $\text{rollNo} \rightarrow \text{studName}; \text{studName} \rightarrow \text{rollNo}\}$

decomposed into

$\text{grades}(\text{rollNo}, \text{course}, \text{grade})$ and $\text{studInfo}(\text{rollNo}, \text{studName})$

is lossless because

$\text{rollNo} \rightarrow \text{studName}$

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

44

A property of lossless joins

$D_1: (R_1, R_2, \dots, R_K)$ lossless decomposition of R wrt F

$D_2: (R_{i1}, R_{i2}, \dots, R_{ip})$ lossless decomposition of R_i wrt $F_i = \Pi_{R_i}(F)$

Then

$D = (R_1, R_2, \dots, R_{i-1}, R_{i1}, R_{i2}, \dots, R_{ip}, R_{i+1}, \dots, R_K)$ is a
lossless decomposition of R wrt F

This property is useful in the algorithm for BCNF decomposition

Prof P Sreenivasa Kumar
Department of CS&E, IITM

45

Algorithm for BCNF decomposition

R – given schema. F – given set of FDs

$D = \{R\}$ // initial decomposition

while there is a relation schema R_i in D that is not in BCNF do

{ let $X \rightarrow A$ be the FD in R_i violating BCNF;

Replace R_i by $R_{i1} = R_i - \{A\}$ and $R_{i2} = X \cup \{A\}$ in D ;

}

Decomposition of R_i is lossless as

$$R_{i1} \cap R_{i2} = X, R_{i2} - R_{i1} = A \text{ and } X \rightarrow A$$

Result: a lossless decomposition of R into BCNF relations

Prof P Sreenivasa Kumar
Department of CS&E, IITM

46

Dependencies may not be preserved (1/2)

Consider the schema: $R(A, B, C)$

with the FDs $F: AB \rightarrow C$ and $C \rightarrow B$

Keys: AB, AC – relation in 3NF (all attributes are prime)

– Relation is not in BCNF as $C \rightarrow B$ and C is not a key

Decomposition given by algorithm: $R_1: CB$ $R_2: AC$

Not dependency preserving as $\Pi_{R_1}(F) = \{C \rightarrow B\}$

$\Pi_{R_2}(F) = \text{trivial dependencies}$

Union of these does not entail $AB \rightarrow C$

All possible decompositions: $\{AB, BC\}, \{BA, AC\}, \{AC, CB\}$

Only the last one is lossless!

Lossless *and* dependency-preserving decomposition doesn't exist!!

Prof P Sreenivasa Kumar
Department of CS&E, IITM

47

Dependencies may not be preserved (2/2)

Consider the schema: townInfo (stateName, townName, distName)

with the FDs $F: ST \rightarrow D$ (town names are unique within a state)

$D \rightarrow S$ (district names are unique across states)

Keys: ST, DT – all attributes are prime

– relation is in 3NF

Relation is not in BCNF as $D \rightarrow S$ and D is not a superkey

Decomposition given by algorithm: $R_1: TD$ $R_2: DS$

Not dependency preserving as $\Pi_{R_1}(F) = \text{trivial dependencies}$

$\Pi_{R_2}(F) = \{D \rightarrow S\}$

Union of these doesn't imply $ST \rightarrow D$

$ST \rightarrow D$ can't be enforced unless we perform a join.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

48

Equivalent Dependency Sets

F, G – two sets of FDs on schema R

F is said to cover G if $G \subseteq F^+$ (equivalently $G^+ \subseteq F^+$)

F is equivalent to G if $F^+ = G^+$ (or, F covers G and G covers F)

Note: To check if F covers G ,

it's enough to show that for each FD $X \rightarrow Y$ in G , $Y \subseteq X_F^+$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

49

Canonical covers or Minimal covers

It is of interest to reduce a set of FDs F into a 'standard' form F' such that F' is equivalent to F .

We define that a set of FDs F is in '*minimal form*' if

- (i) the rhs of any FD of F is a single attribute
- (ii) there are no redundant FDs in F
that is, there is no FD $X \rightarrow A$ in F
s.t $(F - \{X \rightarrow A\})$ is equivalent to F
- (iii) there are no redundant attributes on the lhs of any FD in F
that is, there is no FD $X \rightarrow A$ in F s.t there is $Z \subset X$ for which
 $F - \{X \rightarrow A\} \cup \{Z \rightarrow A\}$ is equivalent to F

Minimal Covers

useful in obtaining a lossless, dependency-preserving
decomposition of a scheme R into 3NF relation schemas

Prof P Sreenivasa Kumar
Department of CS&E, IITM

50

Algorithm for computing a minimal cover

R – given Schema or set of attributes; F – given set of FDs on R

Step 1: $G := F$

Step 2: Replace every fd of the form $X \rightarrow A_1A_2A_3 \dots A_k$ in G by $X \rightarrow A_1; X \rightarrow A_2; X \rightarrow A_3; \dots; X \rightarrow A_k$

Step 3: For each fd $X \rightarrow A$ in G do
 for each B in X do
 if $(G - \{X \rightarrow A\}) \cup \{(X - B) \rightarrow A\}^+ = F^+$ then
 replace $X \rightarrow A$ by $(X - B) \rightarrow A$

Step 4: For each fd $X \rightarrow A$ in G do
 if $(G - \{X \rightarrow A\})^+ = G^+$ then
 replace G by $G - \{X \rightarrow A\}$

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

51

Computing Minimal Covers

Example from Elmasri and Navathe, Database Systems (6th edition)

Determine the minimal cover for $F = \{ B \rightarrow A, D \rightarrow A, AB \rightarrow D \}$

All rhs sets are single attributes. So, Step 2 changes nothing.

If $G = \{ B \rightarrow A, D \rightarrow A, B \rightarrow D \}$, we find that $G^+ = F^+$

In G, since $B \rightarrow D$, $AB \rightarrow AD$ and hence $AB \rightarrow D$

So $AB \rightarrow D$ belongs to G^+ . Hence G covers F

In F, since $B \rightarrow A$, $B \rightarrow AB$.

Since $B \rightarrow AB$, $AB \rightarrow D$, we get $B \rightarrow D$. So $B \rightarrow D$ is in F^+ .

Hence F covers G.

Finally, in G, we find that $B \rightarrow A$ can be obtained for the other two.

Hence, $\{ D \rightarrow A, B \rightarrow D \}$ is a minimal cover for F

Prof P Sreenivasa Kumar
 Department of CS&E, IITM

52

3NF Decomposition Algorithm

R – given Schema; F – given set of FDs on R in *minimal form*

Use BCNF algorithm to get a lossless decomposition $D = (R_1, R_2, \dots, R_k)$

Note: each R_i is already in 3NF (it is in BCNF in fact!)

Algorithm: Let G be the set of FDs not preserved in D

For each FD $Z \rightarrow A$ that is in G

Add relation scheme $S = (B_1, B_2, \dots, B_s, A)$ to D. // $Z = \{B_1, B_2, \dots, B_s\}$

As $Z \rightarrow A$ is in F which is a minimal cover,

there is no proper subset X of Z s.t $X \rightarrow A$. So Z is a key for S!

Any other fd $X \rightarrow C$ on S is such that C is in $\{B_1, B_2, \dots, B_s\}$.

Such fd's do not violate 3NF because each B_i 's is prime attribute!

Thus any scheme S added to D as above is in 3NF.

D continues to be lossless even when we add new schemas to it! (can be shown)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

53

Multi-valued Dependencies (MVDs) and 4NF

studCoursesAndFriends(rollNo, courseNo, frndEmailAddr)

A student enrolls for several courses and has several friends whose email addresses we want to record.

If rows (CS05B007, CS370, shyam@gmail.com) and

(CS05B007, CS376, radha@yahoo.com) appear then

rows (CS05B007, CS376, shyam@gmail.com)

(CS05B007, CS370, radha@yahoo.com) should also appear!

For, otherwise, it implies that having "Shyam" as a friend has something to do with doing course CS370!

Causes a huge amount of data redundancy!

Since there are no non-trivial FD's, the scheme is in BCNF

We say that MVD $\text{rollNo} \twoheadrightarrow \text{courseNo}$ holds

(read as rollNo *multi-determines* courseNo)

By symmetry, $\text{rollNo} \twoheadrightarrow \text{frndEmailAddr}$ also holds

Prof P Sreenivasa Kumar
Department of CS&E, IITM

54

More about MVDs

Consider studCourseGrade(rollNo, courseNo, grade)

Note that $\text{rollNo} \twoheadrightarrow \text{courseNo}$ *does not* hold here even though courseNo is a multi-valued attribute of a student entity

If (CS05B007, CS370, A)
(CS05B007, CS376, B) appear in the data then
(CS05B007, CS376, A)
(CS05B007, CS370, B) will not appear !!

Attribute 'grade' depends on (rollNo, courseNo)

MVD's arise when two or more *unrelated* multi-valued attributes of an entity are sought to be represented together in a scheme.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

55

More about MVDs

Consider

studCourseAdvisor(rollNo, courseNo, advisor)

Note that $\text{rollNo} \twoheadrightarrow \text{courseNo}$ *holds* here

If (CS05B007, CS370, Dr Ravi)
(CS05B007, CS376, Dr Ravi) appear in the data then
swapping courseNo values gives rise to existing rows only.

But, since $\text{rollNo} \rightarrow \text{advisor}$ and $(\text{rollNo}, \text{courseNo})$ is the key, this gets caught in checking for 2NF itself.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

56

MVD Definition

Consider a scheme $R(X, Y, Z)$,

An MVD $X \twoheadrightarrow Y$ holds on R if, for in any instance of R , the presence of two tuples

$(xxx, y_1y_1y_1, z_1z_1z_1)$ and

$(xxx, y_2y_2y_2, z_2z_2z_2)$

guarantees the presence of tuples

$(xxx, y_1y_1y_1, z_2z_2z_2)$ and

$(xxx, y_2y_2y_2, z_1z_1z_1)$

Note that every FD on R is also an MVD!

- the notion of MVD's generalizes the notion of FD's

Prof P Sreenivasa Kumar
Department of CS&E, IITM

57

Alternative definition of MVDs

Consider $R(\underline{X}, Y, \underline{Z})$

Suppose that $X \twoheadrightarrow Y$ and by symmetry $X \twoheadrightarrow Z$

Then, decomposition $D = (XY, XZ)$ of R should be lossless

That is, for any instance r on R , $r = \Pi_{XY}(r) * \Pi_{XZ}(r)$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

58

MVDs and 4NF

An MVD $X \twoheadrightarrow Y$ on scheme R is called *trivial* if either $Y \subseteq X$ or $R = X \cup Y$. Otherwise, it is called *non-trivial*.

4NF: A relation R is in 4NF if it is in BCNF and for every nontrivial MVD $X \twoheadrightarrow A$, X must be a superkey of R .

studCourseEmail(rollNo, courseNo, frndEmailAddr)

is not in 4NF as

rollNo \twoheadrightarrow courseNo and

rollNo \twoheadrightarrow frndEmailAddr

are both nontrivial and rollNo is not a superkey for the relation

Prof P Sreenivasa Kumar
Department of CS&E, IITM

59

Join Dependencies and 5NF

A join dependency (JD) is generalization of an MVD

A JD $JD(R_1, R_2, \dots, R_k)$ is said to hold on schema R if

for every instance $r = *(\Pi_{R_1}(r), \Pi_{R_2}(r), \dots, \Pi_{R_k}(r))$

Here, $R = R_1 \cup R_2 \cup \dots \cup R_k$ and Natural join $*$ is a multi-way join.

A JD is difficult to detect in practice. It occurs in rare situations.

A relational scheme is said to be in 5NF wrt to a set of FDs, MVDs and JDs if it is in 4NF and for every non-trivial $JD(R_1, R_2, \dots, R_k)$, each R_i is a superkey.

Prof P Sreenivasa Kumar
Department of CS&E, IITM

60

Join Dependencies – An Example

Consider the following relation:

studProjSkill(rollNo, skill, project) and the three relations

studSkill(rollNo, skill) // who has what skill

studProj(rollNo, project) // who is interested in what project

skillProj(project, skill) // which project requires what skills

Suppose there is a rule that:

If a student $r1$ has skill $s1$, and $r1$ is interested in project $p1$ and project $p1$ requires skill $s1$ then $(r1, s1, p1)$ *must be* in studProjSkill

In other words, $\text{studProjSkill} = * (\text{studSkill}, \text{studProj}, \text{skillProj})$

Then, we say $\text{JD}(\text{studSkill}, \text{studProj}, \text{skillProj})$ holds

Prof P Sreenivasa Kumar
Department of CS&E, IITM

61

Example - Observations

rollNo	skill
r1	s1
r1	s2

Size $\leq r_s$

rollNo	project
r1	p1
r1	p2

Size $\leq r_p$

project	skill
p1	s1
p2	s3

Size $\leq s_p$

rollNo	project	skill
r1	p1	s1

Size $\leq r_{ps}$

There are no MVDs in 3-column table

#students = r , #projects = p , #skills = s
 $rps \gg rp + sp + rs$

Huge amount of data redundancy exists

Prof P Sreenivasa Kumar
Department of CS&E, IITM

62

Relational DB Design - Approaches

Two Approaches: Bottom-up and Top-down

Bottom-up Approach (aka Synthesis Approach)

- Keep all attributes in a universal relation
- Determine *all* the FDs, MVDs, applicable
- Use the algorithms discussed to decompose the universal relation
- Obtain a design using the algorithms discussed

Drawbacks of the approach

- Difficult to obtain *all* the FDs in a large DB with 100s of attributes
- Algorithms are non-deterministic
- Not popular in practice

Prof P Sreenivasa Kumar
Department of CS&E, IITM

63

Relational DB Design - Approaches

Top-down Approach (aka Analysis Approach)

- Represent Entities/Relationships as relations
 - Group attributes that belong naturally together
- Determine the FDs, MVDs, applicable among attributes
- Analyze the relations individually and also collectively
 - If necessary carry out decomposition to obtain desirable properties
- More popular approach
- Theoretical observations are applicable to both approaches

Prof P Sreenivasa Kumar
Department of CS&E, IITM

64

Database Systems

Algorithms for Relational Algebra Operators and Query Evaluation

Dr P Sreenivasa Kumar

Professor

CS&E Department

I I T Madras

Prof P Sreenivasa Kumar
Department of CS&E, IITM

1

Relational Query Evaluation

- Relational Algebra Operators
 - Select, Project, Join
 - Union, Intersect, Difference
- Grouping and aggregation
 - Sorting
- How to implement these?
- How do indexes help?
- Any other information is helpful?

Prof P Sreenivasa Kumar
Department of CS&E, IITM

2

Selection With Equality Conditions

- Single selection condition $X = c_1$
 - Index on X ? Yes: use the index; No: file scan
- Several conjunctive conditions
 - $X_1 = c_1$ and $X_2 = c_2$ and ... and $X_k = c_k$
 - Index on any X_i ?
 - Yes: Get the records and check other conditions
 - No: File scan
- Several disjunctive conditions
 - Index on any *single* X_i - not helpful
 - Difficult compared to conjunctive case

Prof P Sreenivasa Kumar
Department of CS&E, IITM

3

Predicate Selectivity

- Selectivity s of a condition C -- $0 \leq s \leq 1$
 - (No. of records satisfying C) / (Total no. of records)
 - C_1 : student.dept = "CSE" -- $450 / 8000 = 0.056$
 - C_2 : student.sex = "female" -- $1200 / 8000 = 0.15$
 - C_3 : student.rollNo = "CS10B032" -- $1/8000 = 0.000125$
 - highly selective predicate - very **low** selectivity value
- Conjunction of conditions
 - Choose the one that is *most* selective
 - Get the records and check other conditions
 - Selectivity values (estimates): collect offline

Prof P Sreenivasa Kumar
Department of CS&E, IITM

4

Selectivity Estimation

- Maintained in the DB *catalog*
 - Used by the query optimizer
- Equality conditions involving a key attribute
 - $\text{Selectivity} = 1 / (\text{Total no. of records})$
- Equality conditions involving a non-key attribute
 - $\text{Selectivity} = 1 / (\text{Distinct values of the attribute})$
- Sometimes histograms are also maintained
 - Distinct value or value range -- # of records

Prof P Sreenivasa Kumar
Department of CS&E, IITM

5

Project Operation

- For every record in the operand
 - Access it, take the required attributes values
 - Construct the result record
- Duplicate Elimination
 - Costly
 - Sort or hash based methods are used
- File scan becomes essential
- Apply project after selection, if possible
 - To reduce the input to project

Prof P Sreenivasa Kumar
Department of CS&E, IITM

6

External Sorting

- Sorting a file
 - An often required operation
 - Duplicate elimination, Grouping of records, Join etc
- Merge-sort Principle is used
 - $O(n \log n)$ worst-case complexity for n items
 - Two phases
 - **Sort phase** – repeat: read part of data, sort and write
 - Create many sorted files – called *runs*
 - **Merge phase** – repeat: merge *some* sorted files and write
 - Till only one sorted file is left

Prof P Sreenivasa Kumar
Department of CS&E, IITM

7

Algorithm – Sort Phase

- File: n blocks and Buffer memory: m blocks
- Sort Phase
 - Repeat the following $\lceil n/m \rceil$ times
 - {read the next m blocks; sort in-memory;
write to disk as a single file, called a *run*}
- Number of *runs* $r = \lceil n/m \rceil$
- Complexity: n block reads and n block writes
 - $2n$ block accesses

Prof P Sreenivasa Kumar
Department of CS&E, IITM

8

Algorithm – Merge Phase

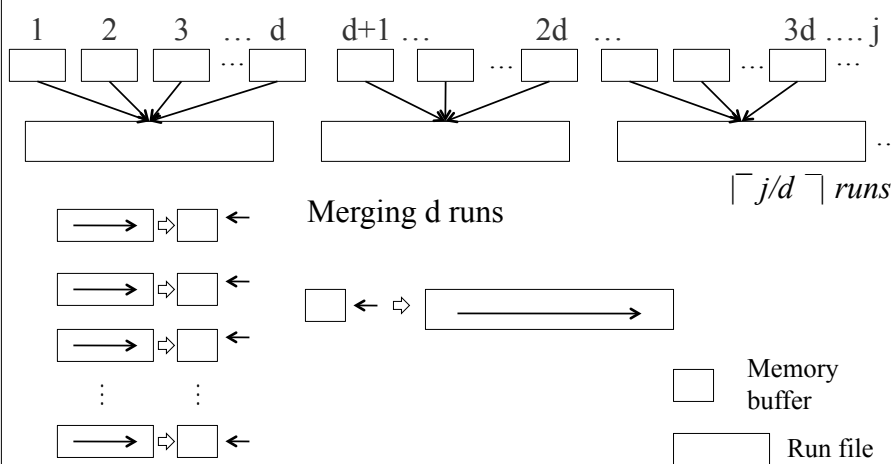
- File: n blocks, Memory Buffers: m (≥ 3) blocks, Runs: r
 - Degree of merging d : $2 \leq d \leq (m-1)$
- Merge Phase: repeat the following $\lceil \log_d r \rceil$ times
 - Reduce j runs to $\lceil j/d \rceil$ runs (Initially, $j = r$)
 - By repeatedly merging d runs at a time to get *one* run
 - Use d buffers, one for each of the next d runs; use one for the result
 - Get one block at a time from each run
 - Merge and write the result to disk – one block at a time
- Complexity: $2n \lceil \log_d r \rceil$
 - Each sub-phase : Entire file gets read and written
- Overall: $(2n + 2n \lceil \log_d r \rceil)$ block accesses

Prof P Sreenivasa Kumar
Department of CS&E, IITM

9

Algorithm – Merge Phase

Reducing j runs to $\lceil j/d \rceil$ runs



Prof P Sreenivasa Kumar
Department of CS&E, IITM

10

Join Processing

- Join – A very important operation
- 2-way join
 - Two files of records, join condition – given
- Multi-way join
- Choice of algorithm depends on ...
 - Sizes of files
 - Primary organization of the files
 - Availability of indices
 - Selectivity of the join condition etc

Prof P Sreenivasa Kumar
Department of CS&E, IITM

11

Nested Loop Join (or block nested loop join)

- Brute force join
- Two data files

for each record x in R do
 for each record y in S do
 check if x, y join ...
- $R : b_1$ blocks, $S : b_2$ blocks, Buffer : m blocks
- Buffer Usage: One block for the result of join
 - One for inner file (say, S); $(m - 2)$ for outer file (R)
- For each *set* of $(m - 2)$ blocks of R read-in, do
 - For each block of S do
 - Read it in, compute join, write to result-block
 - Write the result-block to disk whenever it fills up

Prof P Sreenivasa Kumar
Department of CS&E, IITM

12

Nested Loop Join - Performance

- Two data files
 - $R : b_1$ blocks, $S : b_2$ blocks, Buffer : m blocks
- Outer file : b_1 blocks accesses
- # times inner file blocks accessed: $\lceil b_1 / (m - 2) \rceil$
- Overall: $b_1 + \lceil b_1 / (m - 2) \rceil b_2$
- Or, symmetrically: $b_2 + \lceil b_2 / (m - 2) \rceil b_1$
 - when we have S in the outer loop and R inside
- Which file in the outer loop?

Time for writing the result needs to be added

 - The one with fewer blocks!

Prof P Sreenivasa Kumar
Department of CS&E, IITM

13

Nested Loop Join - Example

- Two data files
 - $R : b_1 = 5600$ blocks, $S : b_2 = 100$ blocks, Buffer : 52 blocks
- If R is used in the outer loop
 - $b_1 + \lceil b_1 / (m - 2) \rceil b_2$
 - $5600 + \lceil 5600 / 50 \rceil * 100 = 16800$ disk ops
- If S is used in the outer loop
 - $100 + \lceil 100 / 50 \rceil * 5600 = 11300$ disk ops
- Assuming 10 msec per disk op
 - It is 168 secs versus 113 secs

Time for writing the result needs to be added

Prof P Sreenivasa Kumar
Department of CS&E, IITM

14

Single Loop Join (or index loop join)

- Two data files
 - R : b_1 blocks, S : b_2 blocks
 - Need to compute **equi-join** with $R.A = S.B$
 - We have index on one of them, say S on B
- For each record x of R read in, do
 - Use the index on B for S
 - Get all the matching records (having $B = x.A$)
- Time taken: $b_1 + |\text{distinct}(R.A)| * h_B(S)$
 - $h_B(S)$ – # of block accesses of the index on B for S

Time for writing the result needs to be added

Prof P Sreenivasa Kumar
Department of CS&E, IITM

15

Join Selection Factor

- Fraction of records in a file that join with records of the other for the given condition
- Consider: professor $\bowtie_{\text{empId} = \text{hod}}$ department
 - Only 5% of professor rows join with department rows
 - 100% of department rows join with professor rows
- Impacts performance of single loop join
 - If indexes are available on both files
 - Loop over records of the file with high join selection factor

Prof P Sreenivasa Kumar
Department of CS&E, IITM

16

Join Selection Factor - Example

- Impacts single loop join performance
 - If indexes are available on both files
- Consider: $\text{professor} \bowtie_{\text{emplId} = \text{hod}} \text{department}$
 - Loop over *professor* records and probe *department* using index on *hod* (option 1) OR
 - Loop over *department* records and probe *professor* using index on *emplId* (option 2)
 - Option 1: 95% probes don't give a match
 - Option 2: All probes give a match
- Option 2 is the right choice

Prof P Sreenivasa Kumar
Department of CS&E, IITM

17

Hash Join

- Consider a 2-way equi-join $R \bowtie_{R.A=S.B} S$
 - Assume that **S fits into memory**
- Use a hash function h
 - Hash the records of S into M buckets using B-values
 - Called the **partitioning** of S
- To compute join result
 - Hash records of R, one by one, using A values
 - Use the **same** M buckets and the **same** hash function h
 - Matching pair of records will hash to same bucket

Prof P Sreenivasa Kumar
Department of CS&E, IITM

18

Partition Hash Join

- Consider a 2-way equi-join $R \bowtie_{R.A=S.B} S$
 - **Neither R nor S fits into the memory**
- Partition Phase: use a hash function h
 - Hash the records of R into m buckets using A-values
 - We get R_1, R_2, \dots, R_m - write them to files
 - Hash the records of S into m buckets using B-values
 - We get S_1, S_2, \dots, S_m - write them to files
 - Goals: ensure that distribution is uniform and
 - At least one of R_i or S_i fit into the memory
- To compute join result: join R_i with S_i only!

Prof P Sreenivasa Kumar
Department of CS&E, IITM

19

Partition Hash Join – Probe Phase

- Probe Phase: Join R_i with S_i for all i
- If one of R_i or S_i fit into the memory
 - Use the idea of hash join again!
 - Hash the smaller of the two into main memory using a *different* hash function, say h_2
 - Read the other file, probe and produce result records
 - Overall cost: $(3(|R|+|S|) + |\text{result}|)$ block accesses
- Else use nested loops join
 - Overall cost: $2(|R|+|S|) + \text{cost of nested loop joins}$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

20

Sort-merge join

- Consider a 2-way equi-join $R \bowtie_{R.A=S.B} S$
- If R is sorted on A, S is sorted on B
 - Merge R and S to get join results
 - Called merge join - - very efficient - - linear
- If one of them is sorted on join attribute
 - Sorting the other and merging may be cost-effective
- Of course, we can
 - Sort R on A, sort S on B and use merge
 - Cost might be high

Prof P Sreenivasa Kumar
Department of CS&E, IITM

21

Set Operations

- Hash based join method
 - Can be adapted to compute Union, Intersect and Difference
- Sort-Merge method
 - Can be adapted to compute Union, Intersect and Difference
- Please study the details!

Prof P Sreenivasa Kumar
Department of CS&E, IITM

22

Query Optimization

- An SQL query - converted to a RA expression tree
- Initial RA expression is re-written
 - Using heuristic and algebraic transformation rules that preserve the meaning of the expression
 - Called algebraic optimization
 - Final RA expression tree is generated
- Cost-based query optimization
 - Cost estimates of *methods* for RA ops are computed
 - Execution plan with least estimated cost is chosen

Prof P Sreenivasa Kumar
Department of CS&E, IITM

23

Heuristic Optimization

- An SQL query - converted to a RA expression tree
- This RA expression tree is to be re-written
- Main heuristic rule
 - Apply *select* and *project* before other operations
 - Reduces the size of intermediate results
 - Reduces the number of fields in the intermediate results
- Make use of relational algebraic laws
 - *Select, project, join, union, intersect* - commutative
 - *Join, union, intersect* - associative
 - There are many more....(Read about them)

Prof P Sreenivasa Kumar
Department of CS&E, IITM

24

Cost-based Optimization

- After initial RA expression tree is re-written using heuristics and algebraic laws....
- Each RA operator
 - Can be evaluated using *many* methods
 - For a method, its *cost function* gives *estimated cost*
 - By taking file sizes, access path costs etc into account
 - Choice made at a node may effect choices at others
- Evaluate different plans based on estimated costs
 - Choose the plan with least estimated cost

Prof P Sreenivasa Kumar
Department of CS&E, IITM

25

Query Optimization – Example

- Obtain the name and phone details of professors who taught the courses taken by student with roll number “CS08B027” in the even semester of 2010
- *select* p.empId, p.name, p.phone
from professor p, teaching t, enrollment e
where e.rollNo = “CS08B027”
 and e.courseId = t.courseId
 and e.sem = “even” and e.year = 2010
 and t.sem = “even” and t.year = 2010
 and p.empId = t.empId

Prof P Sreenivasa Kumar
Department of CS&E, IITM

26

Query Optimization – Example

- Obtain the name and phone details of professors who taught the courses taken by student with roll number “CS08B027” in the even semester of 2010
- Initial RA Expr: $\Pi_{p.empId, p.name, p.phone} (\sigma_{\theta} (p \times t \times e))$
where
 p : professor, t : teaching, e : enrollment
 $\theta = (e.rollNo = \text{“CS08B027” and } e.courseId = t.courseId$
 $\text{and } e.sem = \text{“even” and } e.year = 2010$
 $\text{and } t.sem = \text{“even” and } t.year = 2010$
 $\text{and } p.empId = t.empId)$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

27

Query Optimization – Example

- $\Pi_{p.empId, p.name, p.phone} (\sigma_{\theta} (p \times t \times e))$
 $\equiv \Pi_{p.empId, p.name, p.phone} (\sigma_{\theta_3} (p \times \sigma_{\theta_2} (t) \times \sigma_{\theta_1}(e)))$
- p : professor, t : teaching, e : enrollment
 $\theta_1 = (e.rollNo = \text{“CS08B027” and}$
 $\text{and } e.sem = \text{“even” and } e.year = 2010)$
 $\theta_2 = (t.sem = \text{“even” and } t.year = 2010)$
 $\theta_3 = (p.empId = t.empId \text{ and } e.courseId = t.courseId)$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

28

Query Optimization – Example

- $\Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta} (p \times t \times e))$
- $\equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta_3} (p \times \sigma_{\theta_2} (t) \times \sigma_{\theta_1}(e)))$
- $\equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta_3} (p \times \sigma_{\theta_4} (\sigma_{\theta_2} (t) \times \sigma_{\theta_1}(e))))$

p: professor, t: teaching, e: enrollment

$\theta_1 = (e.\text{rollNo} = \text{"CS08B027"} \text{ and } e.\text{sem} = \text{"even"} \text{ and } e.\text{year} = 2010)$

$\theta_2 = (t.\text{sem} = \text{"even"} \text{ and } t.\text{year} = 2010)$

$\theta_3 = (p.\text{empId} = t.\text{empId})$

$\theta_4 = (e.\text{courseId} = t.\text{courseId})$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

29

Query Optimization – Example

- $\Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta} (p \times t \times e))$
- $\equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta_3} (p \times \sigma_{\theta_2} (t) \times \sigma_{\theta_1}(e)))$
- $\equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta_3} (p \times \sigma_{\theta_4} (\sigma_{\theta_2} (t) \times \sigma_{\theta_1}(e))))$
- $\equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (p \bowtie_{\theta_3} (\sigma_{\theta_2}(t) \bowtie_{\theta_4} \sigma_{\theta_1}(e)))$

$\theta_1 = (e.\text{rollNo} = \text{"CS08B027"} \text{ and } e.\text{sem} = \text{"even"} \text{ and } e.\text{year} = 2010)$

$\theta_2 = (t.\text{sem} = \text{"even"} \text{ and } t.\text{year} = 2010)$

$\theta_3 = (p.\text{empId} = t.\text{empId})$

$\theta_4 = (e.\text{courseId} = t.\text{courseId})$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

30

Query Optimization – Example

$$\begin{aligned}
 & \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta} (p \times t \times e)) \\
 & \equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta_3} (p \times \sigma_{\theta_2}(t) \times \sigma_{\theta_1}(e))) \\
 & \equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta_3} (p \times \sigma_{\theta_4} (\sigma_{\theta_2}(t) \times \sigma_{\theta_1}(e)))) \\
 & \equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (p \bowtie_{\theta_3} (\sigma_{\theta_2}(t) \bowtie_{\theta_4} \sigma_{\theta_1}(e))) \\
 & \equiv (\Pi_{\text{empId}, \text{name}, \text{phone}}(p) \bowtie_{\theta_3} \Pi_{\text{empId}} (\Pi_{\text{courseId}, \text{empId}} \sigma_{\theta_2}(t) \bowtie_{\theta_4} \Pi_{\text{courseId}} \sigma_{\theta_1}(e)))
 \end{aligned}$$

$\theta_1 = (e.\text{rollNo} = \text{"CS08B027"} \text{ and } e.\text{sem} = \text{"even"} \text{ and } e.\text{year} = 2010)$

$\theta_2 = (t.\text{sem} = \text{"even"} \text{ and } t.\text{year} = 2010)$

$\theta_3 = (p.\text{empId} = \text{empId}) \quad \theta_4 = (t.\text{courseId} = e.\text{courseId})$

Prof P Sreenivasa Kumar
Department of CS&E, IITM

31

Cost-based Optimization

$$\begin{aligned}
 & \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta} (p \times t \times e)) \\
 & \equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta_3} (p \times \sigma_{\theta_2}(t) \times \sigma_{\theta_1}(e))) \\
 & \equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (\sigma_{\theta_3} (p \times \sigma_{\theta_4} (\sigma_{\theta_2}(t) \times \sigma_{\theta_1}(e)))) \\
 & \equiv \Pi_{p.\text{empId}, p.\text{name}, p.\text{phone}} (p \bowtie_{\theta_3} (\sigma_{\theta_2}(t) \bowtie_{\theta_4} \sigma_{\theta_1}(e))) \\
 & \equiv (\Pi_{\text{empId}, \text{name}, \text{phone}}(p) \bowtie_{\theta_3} \Pi_{\text{empId}} (\Pi_{\text{courseId}, \text{empId}} \sigma_{\theta_2}(t) \bowtie_{\theta_4} \Pi_{\text{courseId}} \sigma_{\theta_1}(e)))
 \end{aligned}$$

Evaluate costs of using different methods for
the two selections, two joins
and choose the plan with least estimated cost

Prof P Sreenivasa Kumar
Department of CS&E, IITM

32

Query Plan Execution

Intermediate Tables:

Store as files on disk (materialization), if necessary

Use pipelining, as much as possible

Query Types and Optimization

Compiled Queries

Optimization can be done offline

cost of optimization – does not matter

Ad-hoc Queries – Optimization should finish fast

Introduction

Transaction Processing

- A very important component of any online information system / portal
- E-governance applications are in wide-spread use

Transaction

- A logical unit of work to be carried out
- on request by the end-user
 - transfer of specified amount of money from one account to another
 - making a reservation for a journey
 - issue a book of the library to a user

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

1

Assumptions

- DB Server is a single processor system
- 2-Tier architecture is used: DB Server - Client System
- Transactions considered in this module
 - Involve a single DB but not multiple DBs
 - Not nested – a Txn does not initiate another Txn inside it
 - Transactions do not exchange any messages
- E-commerce Transactions
 - Might involve multiple DBs – Merchant, Bank, etc
 - Additional issues need to be considered

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

2

Transaction Processing

Input:

A number of requests for services from the end-users submitted concurrently from several input points

Action:

Carry out the requested services in a *consistent* manner

- no seat on a journey is reserved for more than one person
- amount debited from the party A is credited to party B

Measure:

Maintain a reasonably high throughput

- number of transactions completed per sec

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

3

What is a Transaction?

From the end-user point of view

- a logically sensible/complete piece of work

From the system point of view

- a sequence of database operations
 - read data from tables on the disk,
 - compute/make the updates,
 - write back the data to the disk

Doing one transaction at a time, one-by-one:

- no scope of errors, but slow

Doing multiple transactions at a time

- Scope for error unless done carefully, good throughput

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

4

ACID Properties

- A - Atomicity
- C - Consistency
- I - Isolation
- D - Durability

Important properties to be satisfied by the overall system

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

5

Atomicity

- The work of a transaction should be done entirely as one unit or not performed at all
 - Carrying out some portion of the work leads to inconsistent state of the database
 - For example, Rs.1000/-, to be transferred from A to B
 - Debited from A but not credited to B!
 - system encountered on error and stopped in the middle!
 - can not be allowed; leads to inconsistency
- If only a part of the work is done and error is encountered
 - ensure that effect of the partial work is not reflected in the database
- Responsibility of Recovery Module

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

6

Consistency

- Correctness Assumption:
 - A transaction takes the system from one *consistent* state to another *consistent* state, if executed in *isolation*
 - Responsibility of the application program developer
 - application programs or transaction programs should be carefully developed and thoroughly tested
- During the running of the transaction
 - It is possible that the DB is in an *inconsistent* state
Ex: consider money transfer between two bank accounts

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

7

Isolation

- Let T_1, T_2, \dots, T_n be transactions submitted around the same time
- Isolation: Though the operations of T_i are interleaved with those of others, with respect to T_i ,
 - any other T_j *appears* to have either completed before T_i or started after T_i finished
- The operations of T_j are completely isolated from those of T_i and hence have no effect on T_i
- Responsibility of Concurrency Control module

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

8

Durability

Upon successful completion of a transaction,

System must ensure that its effect is permanently recorded in the database

Also effect of failed transactions is not recorded

Failures

- Transaction program failures
 - internal errors – attempted division by zero etc.
 - transaction aborts
- System crashes
 - power failures etc.

Responsibility of Recovery Module

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

9

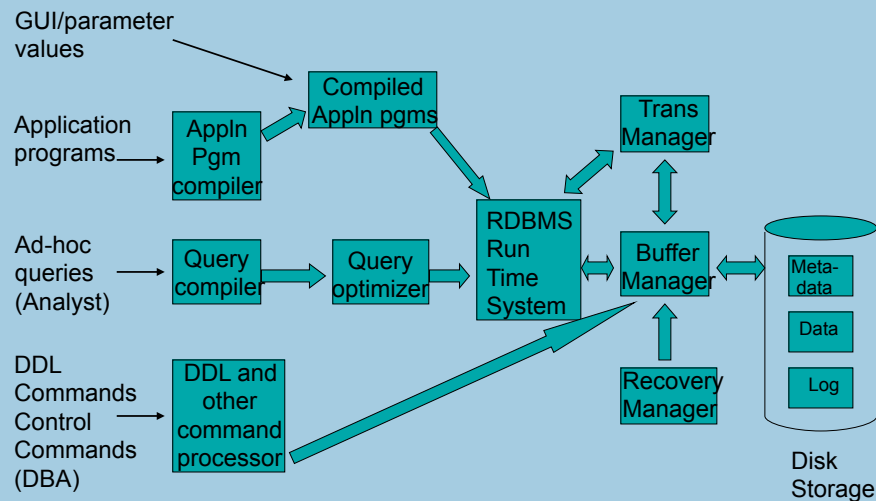
A Note on Transaction Sequencing

- Suppose T_1, T_2, \dots, T_n are transactions submitted around the same time
 - The transactions may not *end* in the same order
- From the system point of view, guaranteeing atomicity and isolation is important
- The ending time of each transaction depends on
 - data items being accessed
 - computation time etc.
 - system's policy for guaranteeing atomicity/ isolation
- From the submitter's point of view
 - when the transaction finishes matters!

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

10

Recall: Architecture of an RDBMS system



Prof P Sreenivasa Kumar
Department of CS&E, IITM

11

Concurrency Control Subsystem

Interleaving the operations of transactions

- essential to achieve good throughput

However, arbitrary interleaving of operations

- leads to inconsistent database

Certain regulated interleaving so that

- two transactions involving the use of same data item don't conflict with each other

Concurrency control subsystem ensures this

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

12

Recovery Manager Subsystem

System Crash

- due to various reasons
- some transactions might have run partially

Transaction Failure

- transaction program error
- transaction was aborted by Concurrency Control Module due to violation of rules

Upon Recovery of system

- effect of partially-run transactions should not be there (atomicity)
- effect of completed transactions must be recorded (durability)

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

13

Recovery Manager and System Log

System Log:

- Details of running transactions are recorded in the log
- Important resource for recovering from crashes
- Always maintained on reliable secondary storage

Log Entries

- beginning of a transaction
- update operation details
 - old value, new value etc.
- ending of transaction
- more details later...

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

14

Transaction Operations

From the DB system point of view, only the read and write operation of a transaction are important

- Other operations happen in memory and don't affect the database on the disk
- Notation:
 - R(X) - transaction Reads item X
 - W(X) - transaction Writes item X
- Database items – denoted by X, Y, Z, ...
 - Would be specified rigorously later

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

15

Transaction Operation - *Commit*

- A transaction issues a *commit* command when
 - it has successfully completed its sequence of operations
 - indicates that its effect *can be* permanently recorded in the db
- Once the DBMS system allows a transaction to commit
 - System is obliged to ensure that the effect of the transaction is permanently recorded in the database
- What exactly happens during *commit*
 - depends on the specific error recovery method used and the specific concurrency control method used
 - will become clear later on in the module

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

16

Transaction Operation - *Abort*

When a transaction issues *Abort*

- indicates that there is some internal error and
- transaction wants to terminate in the middle
 - For instance, a division by zero is being attempted by the txn
 - Or the debit account is going negative if we do money transfer etc

System obligation/ responsibility

- ensure that the partial work done by the transaction has no effect on the database on the disk

What exactly happens during *Abort*

- depends on the specific error-recovery method and the specific concurrency control method adopted by the system

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

17

DB Model for Transaction Processing

- Database
 - Consists of several items – blocks / tuples
 - Granularity of item – affects the algorithms/protocols
 - Usually taken as a block/page
- Transactions operate by exchanging data with DB only
 - They do not exchange messages between them
- Focus on read/write/commit/abort operations
 - Ignore in-memory operations
- Transactions are not nested

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

18

Transaction Operations

- $R_i(X)$ – read op of Txn i , reads db item X
 - disk block having X - copied to buffer page - if reqd
 - required value is assigned to program variable X
- $W_i(X)$ – write op of Txn i , writes db item X
 - disk block having X - copied to buffer page - if reqd
 - update buffer value using program variable X
 - transfer block to disk – immediately or later
- C_i – commit of Txn i
- A_i – abort of Txn i

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

19

Need for Concurrency Control

- If operations of transactions are interleaved arbitrarily
 - Several problems / anomalies arise
 - Can be classified as
 - WR anomalies
 - RW anomalies
 - WW anomalies

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

20

WR Anomalies or Dirty Reads

- Txn T_1 is in progress; updating values in a column
- Txn T_2 reads a value X updated by T_1 , uses it to compute some other quantity, and finishes!
- T_1 for some reason changes X back to its original value!
 - T_2 has read *dirty* data or intermediate data

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

21

RW Anomalies or Unrepeatable Reads

- T_1 has read a value X and intends to read it again before changing it
- Between two reads of X by T_1 ,
 - T_2 reads X , modifies it and finishes
- T_1 reads X the 2nd time and gets a different value!
 - *Unrepeatable read* problem
 - X could be the number of seats available for reservation

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

22

WW Anomalies or Lost Updates

- T_1 reads an item X , reduces it by 10% and wants to write it
- T_2 reads the same X before it was updated by T_1 , increments it by 20% and wants to write it
- Say *write* of T_1 happens and then that of T_2
 - Final value of X is $1.2 * X$!
 - T_1 's Update of X is lost – *lost update* problem

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

23

Transaction Schedules

- Sequence of interleaved operations of a Txn set
 - Ops of a Txn T appear in the same order as in T
- S_1 : $R_1(X) R_2(Y) R_3(X) W_1(X) W_3(X) W_2(Y)$

Time ↓	S1	T1	T2	T3	S	T1	T2	T3
		R(X)				R(X)		
			R(Y)			W(X)		
				R(X)			R(Y)	
		W(X)					W(Y)	
				W(X)				R(X)
			W(Y)					W(X)

- Serial S : $R_1(X) W_1(X) R_2(Y) W_2(Y) R_3(X) W_3(X)$
 - No interleaving

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

24

Serializability

- Serial schedules
 - No interleaving of operations of different txns
 - Do not cause any concurrency problems
 - But performance (throughput) is low
- Serializable Schedules
 - Interleaving of operations happens
 - But, in *some* sense *equivalent* to serial schedules
 - The effect of the interleaving is same as that of *some* serial schedule

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

25

Conflicting Pairs of Operations

- In a schedule, a pair of operations are said to be in *conflict* if
 - The operations belong to two different txns
 - Both the operations deal with the same DB item
 - One of the two ops is a Write operation

- 1) R(X) of T1 conflicts with W(X) of T3
- 2) R(X) of T3 conflicts with W(X) of T1
- 3) W(X) of T1 conflicts with W(X) of T3

S1

T1	T2	T3
R(X)		
	R(Y)	
		R(X)
W(X)		
		W(X)
	W(Y)	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

26

Conflict Equivalence

A schedule S_1 is said to *conflict-equivalent* to S_2 if

The relative order of *any* conflicting pair of operations is the *same* in both S_1 and S_2

$R_1(X) R_2(Y) R_3(X) \underline{W_1(X)} \underline{W_2(X)} W_2(Y)$ (S_1 in the table)

is conflict equivalent to

$R_1(X) R_2(Y) R_3(X) W_2(Y) \underline{W_1(X)} \underline{W_2(X)}$

But is not conflict-equivalent to

$R_1(X) R_2(Y) \underline{W_1(X)} R_3(X) \underline{W_2(X)} W_2(Y)$
(violet pair is out of order)

S1

T1	T2	T3
R(X)		
	R(Y)	
		R(X)
W(X)		
		W(X)
	W(Y)	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

27

Conflict Serializable Schedules

- A schedule is called *conflict serializable* if it is conflict equivalent to *some* serial schedule
- Or, if we can move non-conflicting ops such that
 - The relative order of ops of any Txn is intact and
 - Schedule becomes a serial
- Schedule in picture
 - is *not* conflict-serializable
 - $T1 < T3$ disturbs the 2nd pair and
 - $T3 < T1$ disturbs the 1st and 3rd pairs

S1

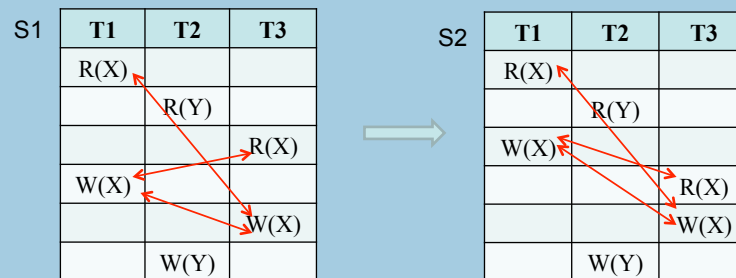
T1	T2	T3
R(X)		
	R(Y)	
		R(X)
W(X)		
		W(X)
	W(Y)	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

28

Conflict Serializable Schedules

Suppose we make a slight change to our example:



S2 is conflict-equivalent to serial schedules

T1,T3,T2 ; T1,T2,T3 and T2,T1,T3

Non-conflicting pairs of ops -- moved to get these serial schedules

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

29

Precedence Graph of a Schedule

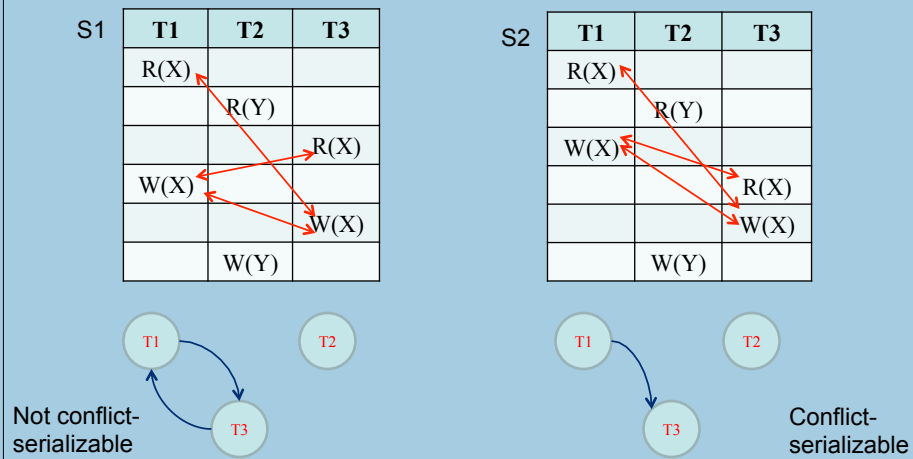
- Precedence graph or serialization graph
- Nodes represent transactions
- A directed arc exists from T_i to T_j if
 - An operation of T_i *precedes* an operation of T_j and *conflicts* with it
- A schedule S is conflict-serializable if and only if the precedence graph of S is *acyclic*
 - Topological sorts of the graph give equivalent serial schedules

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

30

Precedence Graph - Examples

Precedence graph (or serialization graph) examples

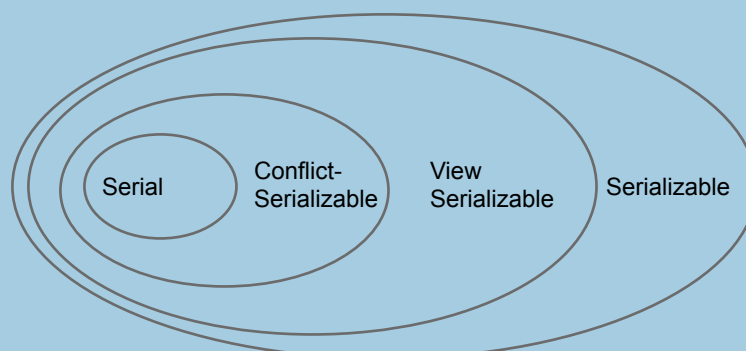


Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

31

Conflict Serializability and Serializability

- Conflict serializability is a sufficient condition for serializability but is not necessary!



Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

32

View Serializability

- A schedule S_1 is view-equivalent to S_2 if any
 - T_i read the initial value of a db item X in S_1 ,
it does so in S_2 also.
 - T_i read a db item X written by T_j in S_1 ,
it does so in S_2 also.
 and,
 - For each db item X , the txn that wrote the final value of X is same in S_1 and in S_2 .
- A schedule is view-serializable
if it is view-equivalent to *some* serial schedule

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

33

Concurrency Control Using Locks

- Assumptions
 - A transaction requests for a lock on a db item X before doing either *read* or *write* on X
 - A transaction unlocks X after it is done with X
- Locks - binary locks are assumed
 - Ensure mutual exclusion
 - At any time, at most one transaction holds a lock on a db item
 - Locking scheduler – ensures above and
 - Keeps track of who holds lock on what item

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

34

Locking and Serializability

T1	T2	T3
L(X),R(X),U(X)		
	L(Y),R(Y),U(Y)	
		L(X),R(X),U(X)
L(X),W(X),U(X)		
		L(X),W(X),U(X)
	L(Y),W(Y),U(Y)	

- Locking alone
 - Does not guarantee serializability
 - Above schedule continues to be non-serializable

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

35

Two Phase Locking (2PL)

- 2PL Protocol
 - All *lock* requests of a transaction precede the first *unlock* request
 - Or a transaction has a *locking* phase followed by an *unlocking* phase
- If all transactions follow 2PL protocol
 - The *resulting* schedules will be conflict-serializable
- A very important and valuable result!

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

36

Why 2PL works?

- S : A schedule of n transactions that follow 2PL
- Let T_i be the transaction
 - that issues the first *unlock* request among all txns
- We will argue that
 - All ops of T_i can be brought to the beginning of S without passing over any conflicting ops
- We get S_1 : (ops of T_i); (ops of other $n-1$ txns)
 - S_1 is conflict-equivalent to S
- Thus we can show that S is conflict-serializable

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

37

Why 2PL works?

- Suppose some op of T_i , say $W_i(X)$, is conflicting with some *preceding* op, say $W_j(X)$, of T_j in S
 - ... $W_j(X)$, ..., $W_i(X)$, ... or we have,
 - ... $W_j(X)$, ..., $U_j(X)$, ..., $L_i(X)$, ..., $W_i(X)$, ...
- As T_i is the *first* txn to issue an unlock, say, $U_i(Y)$,
 - $U_i(Y)$ precedes $U_j(X)$ in S
- So, S could be
 - ... $W_j(X)$, ..., $U_i(Y)$, ..., $U_j(X)$, ..., $L_i(X)$, ..., $W_i(X)$, ...
- Then, T_i is not following 2PL – a contradiction

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

38

Why 2PL works?

- The argument is same for any conflicting pairs of operations involving an operation of T_i
- Thus, for all ops of T_i , there are no conflicting ops that precede it.
- So, beginning with the first op of T_i ,
 - We can swap ops of T_i with the previous ops and bring them all the way to the front of schedule S
- This can be repeated among the remaining $n-1$ txs
- S is conflict-equivalent to a serial schedule

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

39

Possibility of Deadlocks

- Use of basic 2PL
 - Deadlocks may occur; Deadlock detection and resolution is adopted
- To detect
 - A graph called *wait-for* graph is maintained
 - Each running Txn – a node
 - If T_i is waiting to lock an item held by T_j
 - Directed edge from T_i to T_j
- To resolve
 - select Txn in cycles & abort/resubmit them

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

40

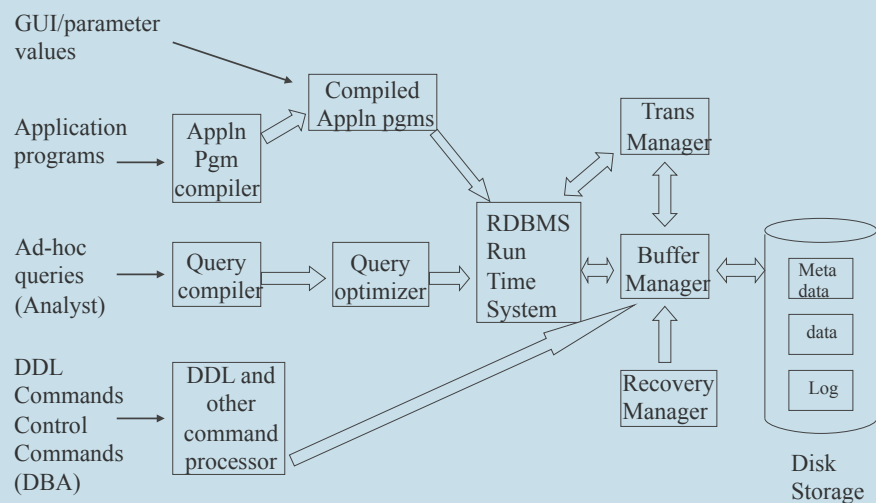
System Failures and Recovery

- Failures
 - Transaction errors – abrupt ending w/o completing
 - CC module may decide to abort a Txn
 - Disk crashes/power failures
- Log – the principal tool for error recovery
 - Sequential file
 - log entries reach the log on disk in same order as they are written (an important assumption)
 - Undo Logs / Redo Logs / Undo-Redo logs
 - Each with a corresponding recovery method

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

41

Architecture of an RDBMS system



Prof P Sreenivasa Kumar
Department of CS&E, IITM

42

Buffer Manager

- DB item – a disk block
- Disk blocks – brought into Memory Buffers
 - Modified by running transactions
 - Written to disk when transaction completes
- We will use more detailed Txn operations
 - Input(A): get disk block with A into buffer
 - Read(A, t): $t := A$; do Input(A) if reqd; t is local var
 - Write(A, t): $A := t$; do Input(A) if reqd
 - Output(A): send block with A from buffer to disk

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

Log Record Entries

- <Start T> -- Txn T has begun
- <End T> -- Txn T has ended
- <Commit T>
 - T has successfully completed its work
 - Changes made by T must appear in the on-disk DB
- <Abort T>
 - T has not successfully completed its work
 - Changes done by T shouldn't be there in the on-disk DB
- Update records – *specific* to the logging method
- Flush Log – Force-write log entries to the disk

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

44

Undo Logging

Update Type Log Record

$\langle T, X, V \rangle$: Txn T has changed the item X and
its *old* value is V

Undo Logging Rules:

- U_1 : If a transaction T modifies db item X, then the log record $\langle T, X, V \rangle$ must be written to disk *before* the new value of X is written to disk.
- U_2 : If a transaction T commits, then its **COMMIT** log record must be written to disk only *after* all db items changed by T have been written to disk, but as soon thereafter as possible.

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

45

Undo Logging

When a Txn T Commits,
DB items flow to disk as below:

for each modified DB item X
 { send update entry $\langle T, X, V \rangle$ to disk
 write item X to disk }
 write $\langle \text{Commit } T \rangle$ to disk

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

46

Undo Logging Example

Txn T is doing money transfer of Rs 100/- from account A to account B

Consistency Requirement: A+B is same before and after the Txn

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

47

Recovery Using Undo Log

Examine Log and Partition Txns into:

Committed Set : Txns for which <Commit T> exists

Incomplete Set : Txns for which <Abort T> exists or
<Commit T> does not exist

Examine Log in the *reverse* direction

For every update record <T,X,V>

T is Committed: **Do nothing**; All is well due to U_2 !

T is Incomplete: Restore value of X on disk as V

- T might have changed some items on disk

- But log entries with old values are on disk due to U_1

Do <Abort T> log entry for each incomplete T & Flush Log

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

48

Undo Logging: Crash Recovery

Crash occurs sometime before the first Flush Log:

DB on disk – unchanged; T - recognized as incomplete;

Log entries of T - unsure if they are on disk; if present - used for “undo”;
no harm! <Abort T> entered; T – resubmitted;

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

49

Undo Logging: Crash Recovery

Crash occurs sometime after the first Flush-Log but before Step 11:

DB on disk - might have changed; T - recognized as incomplete;

Log entries of T - on disk; used for undoing T;

<Abort T> entered; T resubmitted;

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

50

Undo Logging: Crash Recovery

Crash occurs after Step 11, before Step 12: DB on disk - changed;
 <Commit T> - on disk: T - recognized as completed; No action reqd;
 <Commit T> - not on disk: T - recognized as incomplete; Log entries of T used for undoing T; <abort T> entered; T resubmitted;

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

51

Undo Logging: Crash Recovery

Crash occurs after Step 12:
 DB on disk - changed;
 <Commit T> - on disk: T - recognized as completed; No action reqd;

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10	Output(B)	1600	400	1600	400	1600	
11							<commit,T>
12	Flush Log						

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

52

Redo Logging

Undo Logging:

- Cancels the effect of incomplete Txns and
- Ignores the committed Txns
- All DB items to be sent to disk before Txn can commit
- Results in lot of I/O; called FORCE-writing;

Redo Logging:

- Ignores incomplete Txns and
- Repeats the work of Committed Txns
- Update log entry $\langle T, X, V \rangle$: V is the *new* value

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

53

Redo Logging

Redo Logging Rule:

Before modifying any database element X on disk, it is necessary that all log records pertaining to this modification of X, including both the update record $\langle T, X, V \rangle$ and the $\langle \text{COMMIT } T \rangle$ record, *must* appear on disk.

Also called the *write-ahead logging* (WAL) rule

Items flow like this: Update Log records,
the commit entry
and then the changed DB items.

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

54

Redo Logging Example

Txn T is doing money transfer of Rs 100/- from account A to account B

Consistency Requirement: A+B is same before and after the Txn

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,400>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1600>
8							<commit,T>
9	Flush Log						
10	Output(A)	1600	400	1600	400	1500	
11	Output(B)	1600	400	1600	400	1600	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

55

Recovery Using Redo Log

Examine Log and Partition Txns into:

Committed Set : Txns for which <Commit T> exists

Incomplete Set : Txns for which <Abort T> exists or
<Commit T> does not exist

Examine Log in the *forward* direction

For every update record <T,X,V>

T is Incomplete: **Do nothing**; DB on disk has no effects!

T is Committed: Unsure if all the effects of T are on disk

- But log entries with new values are on disk (WAL)

- Redo the change as per the log entry

Do <Abort T> log entry for each incomplete T & Flush Log

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

56

Redo Logging: Crash Recovery

Crash Occurs Before Step 8: <Commit T> not made by T;
T is recognized as incomplete; No action reqd; DB on disk - not
changed (WAL); enter <Abort T>; resubmit T

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,400>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1600>
8							<commit,T>
9	Flush Log						
10	Output(A)	1600	400	1600	400	1500	
11	Output(B)	1600	400	1600	400	1600	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

57

Redo Logging: Crash Recovery

Crash Occurs after Step 8:
<Commit T> - on disk; T is redone with the help of Log entries;
<Commit T> not on disk: T is treated as incomplete; No action reqd;
DB on disk - not changed (WAL); enter <abort T>; resubmit T

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,400>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1600>
8							<commit,T>
9	Flush Log						
10	Output(A)	1600	400	1600	400	1500	
11	Output(B)	1600	400	1600	400	1600	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

58

Redo Logging: Crash Recovery

Crash Occurs after Step 9:

<Commit T> - surely on disk; T is redone with the help of Log entries;
Whether or not T succeeded in writing them!

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,400>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1600>
8							<commit,T>
9	Flush Log						
10	Output(A)	1600	400	1600	400	1500	
11	Output(B)	1600	400	1600	400	1600	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

59

Undo-Redo Logging

Undo Logging:

Cancels the effect of incomplete Txns

All DB items to be sent to disk before Txn can commit

Results in a lot of I/O -- I/O can not be “bunched”

Redo Logging:

Ignores incomplete Txns and

Repeats the work of Committed Txns

Buffer might contain the blocks of several committed

Txns – Buffer utilization might come down!

Undo-Redo Logging:

Better flexibility; uses more detailed logging

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

60

Undo-Redo Logging

Undo-Redo Logging Update Entry:

$\langle T, X, U, V \rangle$:

Txn T has changed the db item X and
its old value is U and new value is V

UR Logging Rule:

Before modifying any database element X on disk, because of changes made by some transaction T, it is necessary that the update record $\langle T, X, U, V \rangle$ appears on disk.

$\langle \text{Commit } T \rangle$ and disk changes – in any order!

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

61

Undo-Redo Logging Example

Txn T is doing money transfer of Rs 100/- from account A to account B
Consistency Requirement: $A+B$ is same before and after the Txn

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							$\langle \text{start } T \rangle$
2	Read(A,m)	500	500		500	1500	
3	$m:=m-100$	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	$\langle T, A, 500, 400 \rangle$
5	Read(B,m)	1500	400	1500	500	1500	
6	$m:=m+100$	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	$\langle T, B, 1500, 1600 \rangle$
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10							$\langle \text{commit}, T \rangle$
11	Output(B)	1600	400	1600	400	1600	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

62

Recovery Using Undo-Redo Logs

Examine Log and Partition Txns into:

Committed Set : Txns for which <Commit T> exists

Incomplete Set : Txns for which <Abort T> exists or
<Commit T> does not exist

Recovery Method:

Redo all committed Txns – order – earliest first

Undo all incomplete Txns – order – latest first

Necessary to do both!

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

63

Undo-Redo Log: Crash Recovery

Crash before <Commit T> is on disk: Txn T – incomplete – undone.

Crash after <Commit T> is on disk: Txn T – completed – redone.

Step	Action	m	Mem-A	Mem-B	Disk-A	Disk-B	Log
1							<start T>
2	Read(A,m)	500	500		500	1500	
3	m:=m-100	400	500		500	1500	
4	Write(A,m)	400	400		500	1500	<T,A,500,400>
5	Read(B,m)	1500	400	1500	500	1500	
6	m:=m+100	1600	400	1500	500	1500	
7	Write(B,m)	1600	400	1600	500	1500	<T,B,1500,1600>
8	Flush Log						
9	Output(A)	1600	400	1600	400	1500	
10							<commit,T>
11	Output(B)	1600	400	1600	400	1600	

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

64

Issue of Dirty Data and Commits

- Consider Txns T and S:
 - T has modified a db item X and it is doing some more work
 - Meanwhile, S has read X, completed its work
 - Suppose S is allowed to Commit
 - Now, T has an internal error and decides to Abort
 - S has read 'dirty' data
 - But S can't be *undone* as it was allowed to *commit*
- DB got into a trouble!

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

65

Recoverable Schedules

- A schedule S is called ***recoverable*** if no Txn T in S *commits* until all the Txns, that have written an item T reads, have committed
 - T needs to wait till each of the Txn from which it has read completes.
 - If all commit, then T can go ahead and commit
 - If at least one such Txn aborts, T also has to abort
 - Cascading Aborts/Rollbacks may occur
- Recoverability is an essential requirement!!

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

66

Recoverability vs Serializability

- Orthogonal concepts
- Both are important for a Transaction System!
- It is possible that a recoverable schedule is not conflict-serializable
 - Recoverability defn has no restrictions on locking
- It is also possible that a serializable schedule is not recoverable
 - Serializability definition has no restriction on committing

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

67

Cascadeless Schedules

- A recoverable schedule
 - may result in cascading rollbacks or aborts
- A schedule is called *cascadeless* or ACR (avoiding cascading rollbacks) if
 - in the schedule, Txns read only values written by *committed* Txns
- Every ACR schedule is recoverable
 - Such a Txn surely commits only after all the Txns it has read from commit; in fact it does not read values written by uncommitted Txns.

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

68

Strict Schedules

- A schedule is called *strict* if in the schedule
 - a Txn neither reads nor writes an item X until the last Txn that writes X has terminated
- Strict 2PL:
 - A Txn must not release any write locks until the Txn has either committed or aborted and the commit or abort log record has been written to disk
 - Results in strict schedules
 - Strict schedules are *cascadeless* and *serializable*

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

69

Transactions in SQL

- Important parameters of Transactions in SQL:
 - Can be set for each transaction
 - Access-Mode:
 - Read-Only; Read-Write (default)
 - Isolation Level: Default is *serializable*
 - Other lower isolation levels are also available
 - Meant for running transactions that collect statistics
 - For isolation level “uncommitted” – Read-Only Txns only
- Each transaction ends with Commit or Rollback

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

70

Transaction isolation levels

Level	Dirty Reads	Unrepeatable Reads	Phantoms
Read Uncommitted	May Be	May Be	May Be
Read Committed	No	May Be	May Be
Repeatable Read	No	No	May Be
Serializable	No	No	No

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

71

Phantom Records

- Phantom records problem:
 - Txn T1 has selected a set of tuples based on a certain condition C, say “student.Dept = 5”
 - And is working with them, say get max(marks)
 - Txn T2 updated the DB with a new tuple that satisfies C after T1 started
 - Can cause T1 to be incorrect
 - Such rows are called *phantom* rows
 - Come into picture out of the blue...
 - “index locking” needs to be adopted

Prof P Sreenivasa Kumar,
Department of CS&E, IITM.

72