SFWRENG 3DB3
DATABASE SYSTEMS
FALL 2021

Fei Chiang (fchiang@mcmaster.ca)

1

Administration

2

- Instructor: Fei Chiang
- Course website: MS Teams + Avenue
  - MS Teams: Lecture recordings, tutorial slides, lecture slides, assignments, policies
  - Avenue: Assignment submission and grading
- Lectures:
  - Mon/Wed/Thurs at 1:30pm-2:20pm
- Tutorials:
  - (T01) Tues: 1:30-2:20pm
  - (T02) Mon: 12:30-1:20pm
  - (T03) Wed: 11:30am – 12:20pm
  - **Note:** Tutorials start next week (week of Sept 13th)

2

Admin (cont'd)

3

- Teaching Assistants:
  - Morteza Alipour Langouri (alipoum@mcmaster.ca)
  - Levin Noronha (noronl@mcmaster.ca)
  - Lucia Cristiano (cristial@mcmaster.ca)
  - Office hours: listed on course info sheet
- Textbook: Database Management Systems (3rd edition) by R. Ramakrishnan, J. Gehrke.
- Bookstore:
  - https://campusstore.mcmaster.ca/cgi-mcm/ws/txsub.pl?wsTERMG1=214&wsDEPTG1=SFWRENG&wsCOURSEG1=3DB3&wsSECTIONG1=DAY%20C01&crit_cnt=1

3

Grading

4

	Asg1	Asg2	Asg3	Total
Assignments	12%	14%	14%	40%
Midterm		20%		20%
Final Exam		40%		40%

Midterm: Thurs Oct. 21, 2021 during lecture time.

4

Lectures, Tutorials, Office Hours

5

- Microsoft Teams
- Channel for:
  - Lectures
  - Each tutorial
  - Each TA and instructor office hour
- Q & A

5

Assignments

6

- Will be posted on Avenue
- Submit through Avenue
- Late policy:
  - Marked with a late penalty of 20% per day
  - No assignments will be accepted beyond 5 days past the due date
  - Do not wait until deadline to raise problems
- Re-marking
  - Within 7 days of returning the assignment

6

Plagiarism

7

- You are encouraged to talk to your fellow students, but submitted work must be based on your own ideas and conclusions.
- Plagiarism and cheating are serious academic offenses, and will be handled accordingly.
- When you submit assignments with your name on it, you are certifying that you have completed the work for that assignment yourself.
- Will use Avenue for assignment submission

7

Questions

8

- If something is unclear, please do ask questions in class!
- E-mail is preferred contact method (fchiang@mcmaster.ca)
- Office hours: Mon 11:30 – 12:30pm
- Feedback is encouraged. If something is concerning you, please let me know early!

8

Topics

9

- Relational Model
- E-R Model
- SQL
- Views, Indexes, Constraints
- Relational Algebra
- Database Design
- Transactions
- Concurrency
- Advanced Topics
  - Data Mining
  - Sustainability

We will be using IBM DB2

"Accessing DB2 Servers" in Avenue

9

10

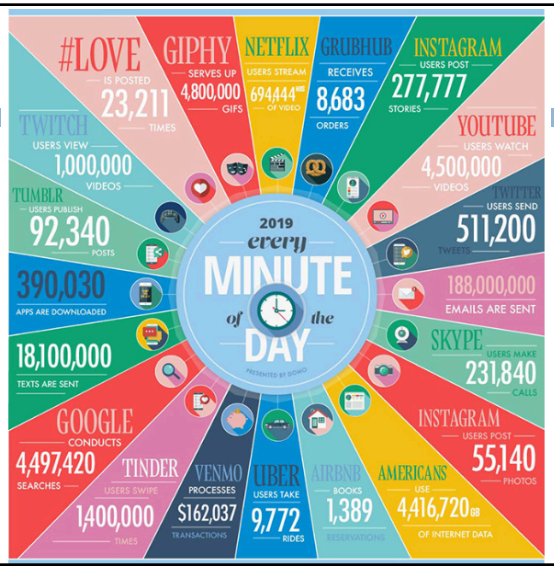
2.5M Terabytes
(or equivalently 2.5 quintillion bytes)

Amount of data generated every day

5M Terabytes of data generated up to 2003

10

11



2019

11

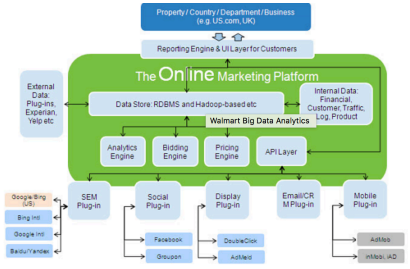
Walmart

13

The analysis covers millions of products and 100's of millions customers from different sources.

Walmart observed a significant 10% to 15% increase in online sales for $1 billion in incremental revenue.

Walmart Labs analyses every clickable action on Walmart.com–
1)What consumers buy in-store and online?
2) What is trending on Twitter?
3) Local events such as San Francisco giants winning the World Series?
4) How local weather deviations affect the buying patterns?



Predictive Analytics: 2.5 petabytes/hr of data from 1M customers, and product interactions worldwide. Weather, economic, telecom data, gas prices, local events…

"I love Salt"

13

What Is a Database *System*?

14

- Database:
  a very large, integrated collection of data.
- Models a real-world *enterprise*
  - Entities (e.g., teams, games)
  - Relationships
    (e.g., Barack Obama *received* the Nobel Peace Prize)
- A *Database Management System (DBMS)* is a software system designed to store, manage, and facilitate access to data.

Credit: Renee J. Miller

14

Is the WWW a DBMS?

15

- Fairly sophisticated search available
  - crawler *indexes* pages on the web
  - Keyword-based search for pages
- But, currently
  - data is mostly unstructured and untyped
  - search only:
    - can't modify the data
    - can't get summaries, complex combinations of data
  - few guarantees provided for freshness of data, consistency across data items, fault tolerance, …
  - Web sites (e.g. e-commerce) typically have a DBMS in the background to provide these functions.

15

Search vs. Query

16

- What if you wanted to find out how to donate to help victims of hurricane Dorian?
- Search for "hurricane Dorian donations" in your search engine.

Hurricane Dorian: Disaster Relief & Donations | American Red Cr…
https://www.redcross.org › donate › hurricane-dorian-donations
Help people affected by Hurricane Dorian. Your donation enables the Red Cross to prepare for, respond to and help people recover from this disaster. ... Contributions to the American National Red Cross are tax-deductible to the extent permitted by law.

How You Can Help the Bahamas Recover From Hurricane Dorian
https://www.miaminewtimes.com › news › miami-groups-launch-fundraise...
6 hours ago - Donations are also being accepted online at the Hurricane Dorian Relief Fund organized by The Smile Trust, a Miami-based nonprofit that ...

Here's how to donate, help Bahamas recover from Dorian - WPLG
https://www.local10.com › weather › hurricane › heres-how-to-donate-hel...
17 mins ago - Videos show Category 5 Hurricane Dorian's fury in Bahamas ... has set up an online fund to donate money for their hurricane relief efforts.

Donations begin to help Bahamians post Hurricane Dorian - YouT…
https://www.youtube.com › watch
9 hours ago - The South Florida community is beginning to collect relief supplies for the people of the Bahamas as Hurricane Dorian continues to cause ...

Hurricane Dorian : Charity Navigator
https://www.charitynavigator.org › ...
Hurricane Dorian, the first major hurricane of the 2019 storm season, ... Designated donations made from this page will be applied to charity programs per each ...

Hurricane Dorian: How You Can Help - Donate Blood - The Blood…
https://thebloodconnection.org › hurricane
3 hours ago - As evacuations begin for those in the path of Hurricane Dorian, The ... the areas where TBC is unable to collect donations during the storm.

Hurricane Dorian: Unused supplies can be donated to charitable …
https://www.news-press.com › story › weather › hurricane › 2019/09/01 › h...
23 hours ago - With this region out of the cone, residents are urged to donate some of ... Hurricane Dorian: Surplus of storm supplies provides opportunity to ...

Investigate before you donate to Hurricane Dorian disaster relief |…
komonews.com › news › consumer › investigate-before-you-donate-to-hu...
3 days ago - Those of us who are not in the path of a major hurricane can only imagine the fear and hope as we watch people prepare for the worst. We've ...

16

Search

17

- Based on keyword matching
  - Our search matches warnings about donations to legitimate places
  - Ranking of results
  - Popularity or reputation
- Web documents
  - Limited structure

17

Search vs. Query

18

- "Search" : returns stored documents "as-is"

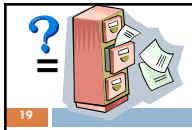
08.29.2019 | Emergency Management

Hurricane Dorian is on the way and Verizon is ready

We're ready to support V Teamers

We're also ready to help employees in times of need through the VtoV Employee Relief Fund. The charity provides grants for Verizon employees displaced from their homes due to a natural or personal emergency - such as fire, flood, severe weather or domestic violence. VtoV is entirely supported by employee donations and the Verizon Foundation's generous matching gift program, with 100% of all donations going to help employees.

18



Is a File System a DBMS?

19

- Thought Experiment 1:
  - You and your project partner are editing the same file.
  - You both save it at the same time.
  - Whose changes survive?

A) Yours B) Partner's C) Both D) Neither E) ???

- **Thought Experiment 2:**
  - –You're updating a file.
  - –The power goes out.
  - –Which of your changes survive?

Q: How do you write programs over a subsystem when it promises you only "???" ?

A: Very, very carefully!!

A) All B) None C) All Since last save D) ???

19

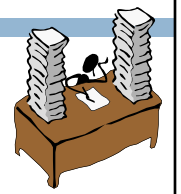
Why Use a DBMS?

- Data independence and efficient access.
- Reduced application development time.
- Data integrity and security.
- Concurrent access, recovery from crashes.

20

Why Study Databases??

21



- Shift from *computation* to *information*
  - always true for corporate computing
  - Web made this point for personal computing
  - more and more true for scientific computing
- Strong need for DBMS
  - Corporate: retail swipe/clickstreams, "customer relationship mgmt", "supply chain mgmt", "data warehouses", Big Data, etc.
  - Scientific: digital libraries, Human Genome project, Sloan Digital Sky Survey, physical sensors
- DBMS encompasses much of CS in a practical discipline
  - OS, languages, theory, machine learning, logic
  - Yet traditional focus on real-world apps

21

What's the intellectual content?

22

- representing information
  - data modeling
- languages and systems for querying data
  - complex queries with real *semantics*
  - over massive data sets
- *concurrency control* for data manipulation
  - controlling concurrent access
  - ensuring *transactional semantics*
- reliable data storage
  - maintain data semantics even if you pull the plug

22

Describing Data: Data Models

23

- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using a given data model.
- The *relational data model* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Every relation has a schema, which describes the columns, or fields.

23

Data Independence

24

- Applications insulated from how data is structured and stored.
- Logical data independence: Protection from changes in *logical* structure of data.
- Physical data independence: Protection from changes in *physical* structure of data.
- Q: Why is this particularly important for DBMS?

> Because rate of change of DB applications is slow. More generally:
> *dapp/dt << dplatform/dt*

24

Describing Data: Data Models

1

- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using a given data model.
- The *relational data model* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Every relation has a schema, which describes the columns, or fields.

1

Data Independence

2

- Applications insulated from how data is structured and stored.
- Logical data independence: Protection from changes in *logical* structure of data.
- Physical data independence: Protection from changes in *physical* structure of data.
- Q: Why is this particularly important for DBMS?

Because rate of change of DB applications is slow. More generally: *dapp/dt << dplatform/dt*

2

Concurrency Control

3

- Concurrent execution of user programs: key to good DBMS performance.
  - Disk accesses frequent
  - Keep the CPU working on several programs concurrently.
- Interleaving actions of different programs: trouble!
  - e.g., account-transfer & print statement at same time
- DBMS ensures such problems don't arise.
  - Users/programmers can pretend they are using a single-user system. (called "Isolation")
  - Thank goodness! Don't have to program "very, very carefully".

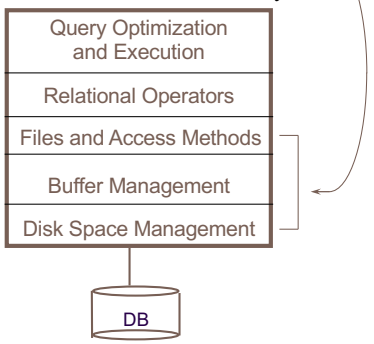
3

Structure of a DBMS

4

These layers must consider concurrency control and recovery

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- Each system has its own variations.



4

So Why Don't We Always Use a DBMS?

5

- Expensive/complicated to set up & maintain
- This cost & complexity must be offset by need
- General-purpose, not suited for special-purpose tasks (e.g. text search!)

5

Summary

6

- DBMS used to maintain, query large datasets.
  - can manipulate data and exploit *semantics*
- Other benefits include:
  - Data independence,
  - quick application development,
  - data integrity and security,
  - recovery from system crashes,
  - concurrent access.
- Levels of abstraction provide data independence
  - Key when $d_{app}/dt << d_{platform}/dt$
- In this course we will explore:
  - How to be a sophisticated user of DBMS technology
  - What goes on inside the DBMS

6

ENTITY-RELATIONSHIP MODEL

7

Overview of Database Design

- Conceptual design:
  - What are the *entities* and *relationships* in the enterprise?
  - What information about these entities and relationships should we store in the database?
  - What are the *integrity constraints* or *business rules* that hold?

8

Purpose of E/R Model

- The Entity/Relationship (E/R) model allows us to sketch database schema designs.
  - Includes some constraints
- Schema designs are pictures called *entity-relationship diagrams*.
- Later: convert E/R designs to relational DB designs.

Credit: Renee J. Miller

9

Framework for E/R

- Design is a necessity.
- Management know they want a database, but they don't know what they want in it.
- Sketching the key components is an efficient way to develop a working database.

10

Entity Sets

11

- *Entity* = "thing" or object.
- *Entity set* = collection of similar entities.
  - Similar to a class in object-oriented languages.
- *Attribute* = property of an entity set.
  - Attributes are simple values, e.g. integers or character strings, not structs, sets, etc.
  - Each attribute has a *domain*.

11

E/R Diagrams

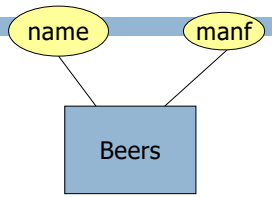
12

- In an entity-relationship diagram:
  - Entity set = rectangle.
  - Attribute = oval, with a line to the rectangle representing its entity set.
  - Notation varies: some textbooks represents attributes within the (entity) rectangle

12

Example

13



- Entity set Beers has two attributes, name and manf (manufacturer).
- Each Beers entity has values for these two attributes, e.g. (Bud, Anheuser-Busch)

13

Relationships

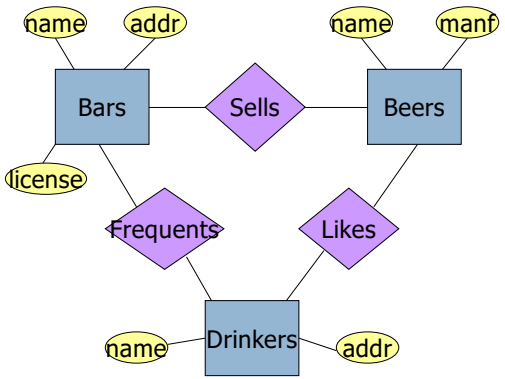
14

- A relationship connects two or more entity sets.
- It is represented by a diamond, with lines to each of the entity sets involved.

14

Example: Relationships

15



Bars sell some beers.

Drinkers like some beers.

Drinkers frequent some bars.

15

Relationship Set

16

- The current "value" of an entity set is the set of entities that belong to it.
  - Example: the set of all bars in our database.
- The "value" of a relationship is a *relationship set*, a set of tuples with one component for each related entity set.

16

Example: Relationship Set

17

- For the relationship Sells, we might have a relationship set like:

Bar	Beer
Joe's Bar	Bud
Joe's Bar	Miller
Sue's Bar	Bud
Sue's Bar	Pete's Ale
Sue's Bar	Bud Lite

17

Multiway Relationships

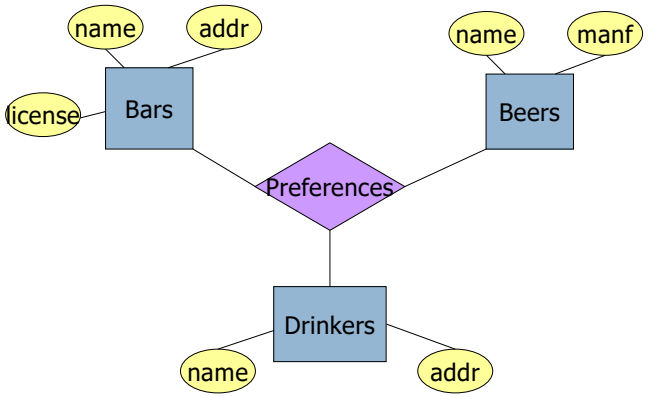
18

- Sometimes, we need a relationship that connects more than two entity sets.
- Suppose that drinkers will only drink certain beers at certain bars.
  - Our three binary relationships Likes, Sells, and Frequents do not allow us to make this distinction.
  - But a 3-way relationship would.

18

Example: 3-Way Relationship

19



19

A Typical Relationship Set

20

Bar	Drinker	Beer
Joe's Bar	Ann	Miller
Sue's Bar	Ann	Bud
Sue's Bar	Ann	Pete's Ale
Joe's Bar	Bob	Bud
Joe's Bar	Bob	Miller
Joe's Bar	Cal	Miller
Sue's Bar	Cal	Bud Lite

20

Many-Many Relationships

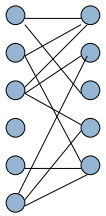
21

- Focus: binary relationships, such as Sells between Bars and Beers.
- In a *many-many* relationship, an entity of either set can be connected to many entities of the other set.
  - E.g., a bar sells many beers; a beer is sold by many bars.

21

In Pictures:

22



many-many

Note: each line is an instance of the binary relationship

22

Many-One Relationships

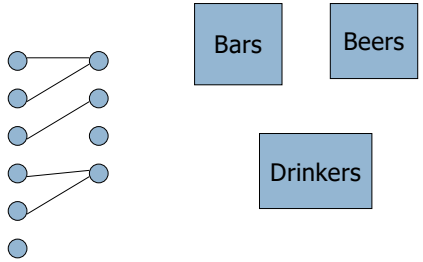
23

- Some binary relationships are *many -one* from one entity set to another.
- Each entity of the first set is connected to at most one entity of the second set.
- But an entity of the second set can be connected to zero, one, or many entities of the first set.

23

In Pictures:

24



many-one

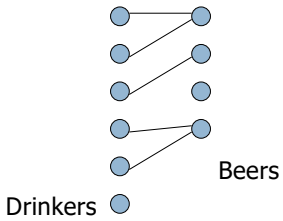
(Partial) Function on entity set

24

Example: Many-One Relationship

25

- Favourite, from Drinkers to Beers is many-one.
- A drinker has at most one favourite beer.
- But a beer can be the favorite of any number of drinkers, including zero.

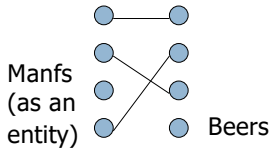


25

One-One Relationships

26

- In a *one-one relationship*, each entity of either entity set is related to at most one entity of the other set.
- Example: Relationship Best-seller between entity sets Manfs (manufacturer) and Beers.
  - A beer is the best seller for 0 or 1 manufacturers, and no manufacturer can have more than one best-seller (assume no ties).



26

Representing "Multiplicity"

27

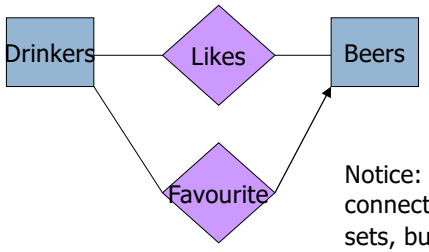
- Show a many-one relationship by an arrow entering the "one" side.
  - "at most one"
- Show a one-one relationship by arrows entering both entity sets.

Rounded (open) arrow = "exactly one," i.e., each entity of the first set is related to exactly one entity of the target set.

27

Example: Many-One Relationship

28

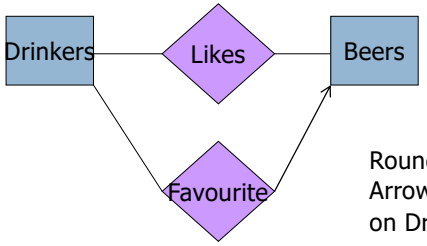


Notice: two relationships connect the same entity sets, but are different.

28

Example: Many-One Relationship

29



Rounded (open) Arrow = total function on Drinkers

29

Example: One-One Relationship

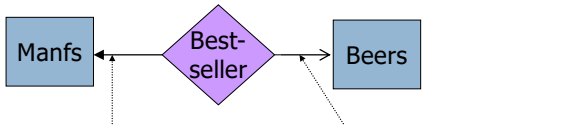
30

- Consider Best-seller between Manfs and Beers.
- Some beers are not the best-seller of any manufacturer
- But a beer manufacturer has to have a best-seller.

30

In the E/R Diagram

31



A beer is the best-seller for 0 or 1 manufacturer.

A manufacturer has exactly one best seller.

31

Participation Constraints

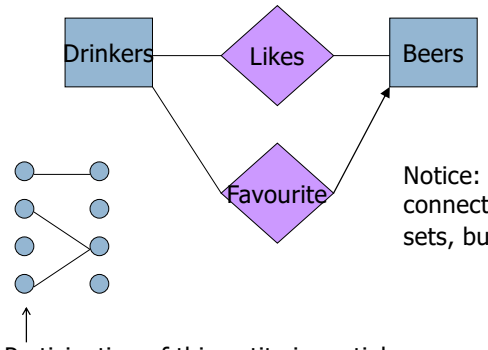
32

- Does every student have to take a course?
  - If so, this is a *participation constraint*: the participation of Students in Enrolled is said to be *total* (vs. *partial*).
  - Every *sid* value in Students table must appear in a row of the Enrolled table (with a non-null *sid* value!)
- Textbook notation: total participation represented by a thick (bolded) line originating from entity

32

Example: Many-One Relationship

33



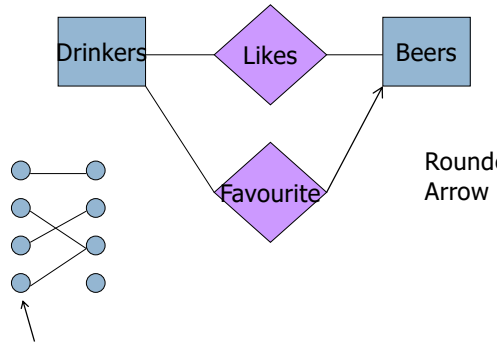
Notice: two relationships connect the same entity sets, but are different.

Participation of this entity is partial

33

Example: Many-One Relationship

34



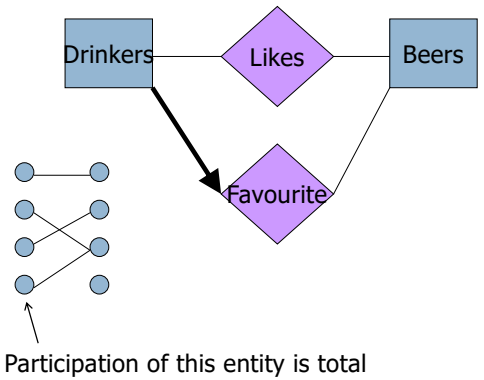
Rounded (open)
Arrow = total function

Participation of this entity is total

34

Alternative (Textbook) Notation

35



35

Announcements

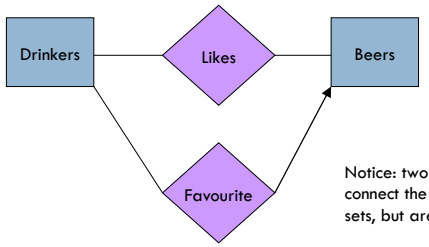
1

- Lecture topics and textbook readings will be added to Avenue calendar
- Looking for 2 undergrad TAs for Winter 2022
  - Will be interviewing post-midterm
  - Expected good performance in **this** course ☺

1

Example: Many-One Relationship

2

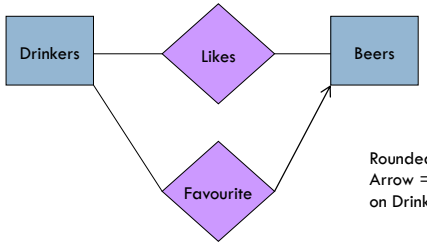


Notice: two relationships connect the same entity sets, but are different.

2

Example: Many-One Relationship

3



Rounded (open)
Arrow = total function on Drinkers

3

Example: One-One Relationship

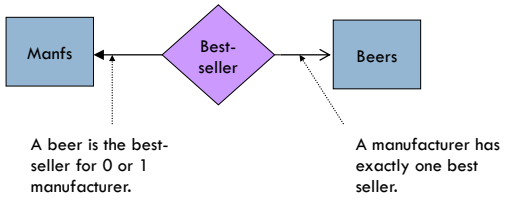
4

- Consider Best-seller between Manfs and Beers.
- Some beers are not the best-seller of any manufacturer
- But a beer manufacturer has to have a best-seller.

4

In the E/R Diagram

5



5

Participation Constraints

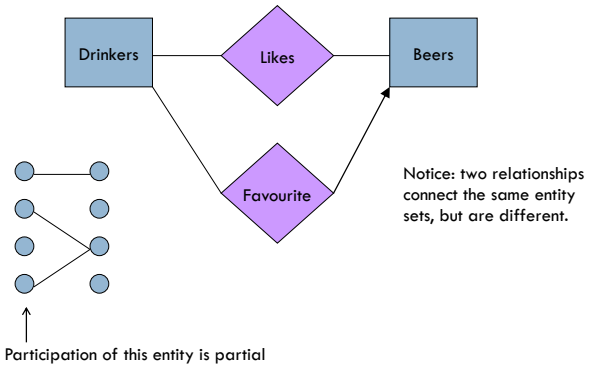
6

- Does every student have to take a course?
  - If so, this is a *participation constraint*: the participation of Students in Enrolled is said to be *total* (vs. *partial*).
  - Every *sid* value in Students table must appear in a row of the Enrolled table (with a non-null *sid* value!)
- Textbook notation: total participation represented by a thick (bolded) line originating from entity

6

Example: Many-One Relationship

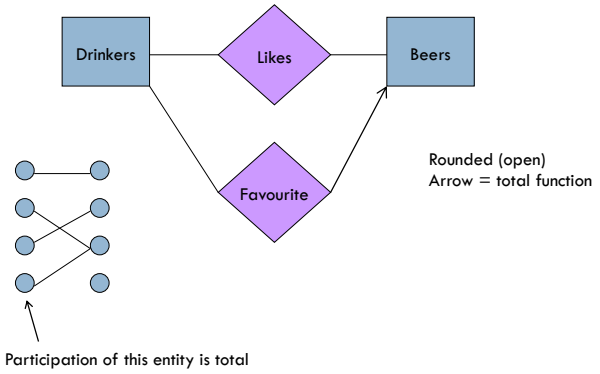
7



7

Example: Many-One Relationship

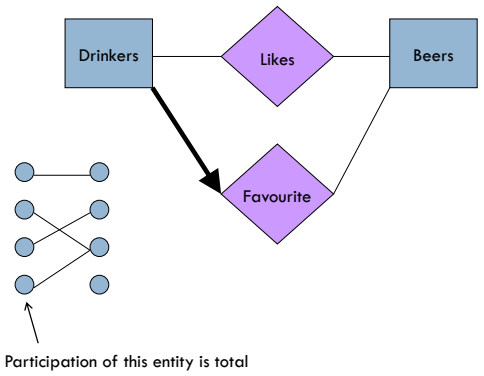
8



8

Alternative (Textbook) Notation

9

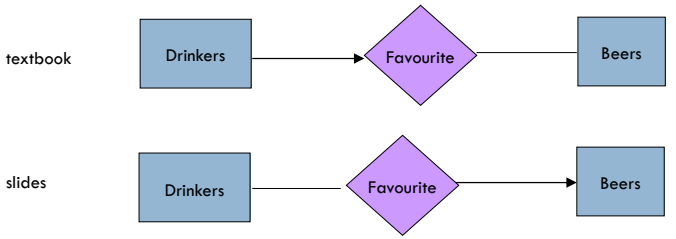


9

Notation

10

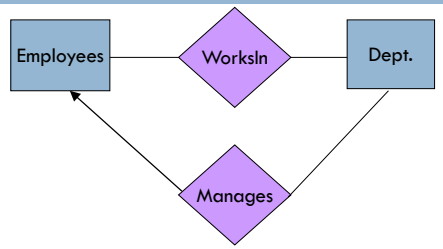
- Be consistent with your chosen notation!



10

Key Constraints

11



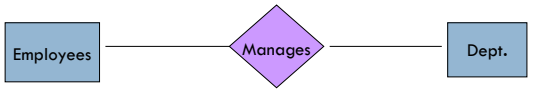
- Many-many: "An employee can work in many depts, and a dept. can have many employees
- One-many: A dept has at most one manager, and employees can manage many departments

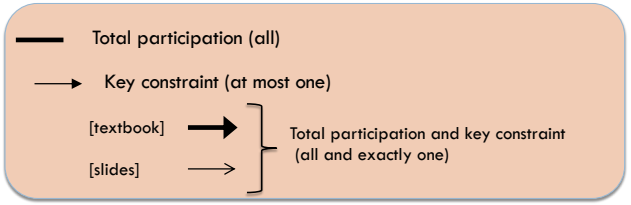
11

Participation Constraints

12

- Does every dept. have to have a manager?
  - If yes, then every dept. must appear in the manages relation: total participation (vs. partial)





12

Attributes on Relationships

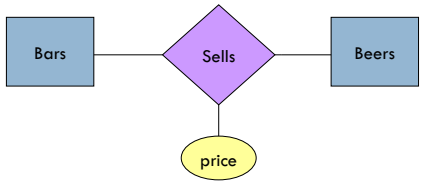
13

- Sometimes it is useful to attach an attribute to a relationship.
- Think of this attribute as a property of tuples in the relationship set.

13

Example: Attribute on Relationship

14



Price is a function of both the bar and the beer, not of one alone.
E.g., "The price of Miller beer at Joe's bar"

14

RELATIONAL DATA MODEL

Fei Chiang (fchiang@mcmaster.ca)

15

Describing Data: Data Models

16

- A *data model* is a collection of concepts for describing data.
- A *schema* is a description of a particular collection of data, using a given data model.
- The *relational model of data* is the most widely used model today.
  - Main concept: *relation*, basically a table with rows and columns.
  - Use tables to represent data and relationships
  - Every relation has a *schema*, which describes the columns, or attributes.

Acknowledgement: Renee J. Miller

16

Relational Model

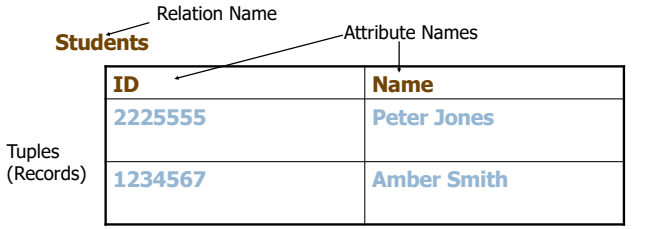
17

- Proposed by Edgar. F. Codd in 1970 as a data model which strongly supports data independence.
- Made available in commercial DBMSs in 1981 -- it is not easy to implement data independence efficiently and reliably!
- It is based on (a variant of) the mathematical notion of ***relation***.
- Relations are represented as tables.

17

A relation is a table

18



The set of permitted values for an attribute is called the attribute ***domain***.
E.g., domain(ID) = {2225555, 1234567}.

18

Relational Data Model

19

- *Relation schema* = relation name and attribute list.
  - Optionally: types of attributes. For example:
  - *Students(id, name)*
  - *Students(id: string, name: string)*
- *Relation* = set of tuples conforming to schema
  - Example:
    - { (2225555, Peter Jones), (1234567, Amber Smith), …}
- *Database* = set of relations.
- *Database schema* = set of all relation schemas in the database.

19

Why Relations?

20

- Very simple model.
- *Often* matches how we think about data.
- Abstract model that underlies SQL, one of the most important database languages today.

20

Relations are Unordered

21

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- E.g., *instructor* relation with unordered tuples

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

21

Database

22

- Information about an enterprise is broken up into parts

  instructor
  student
  advisor

- Bad design:

  univ (instructor_ID, name, dept_name, salary, student_Id, ..)

  results in
  - repetition of information (e.g., two students have the same instructor)
  - the need for NULL values (e.g., represent an student with no instructor)
- Normalization theory deals with how to design "good" relational schemas

22

Database Schemas in SQL

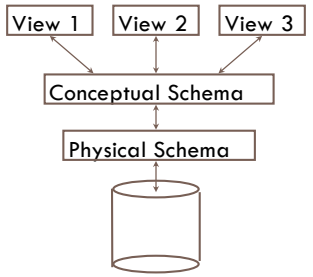
23

- SQL is primarily a query language, for getting information from a database.
- But SQL also includes a *data-definition* component for describing database schemas.

23

Levels of Abstraction

- Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.



- *Schemas are defined using DDL (data definition language);*
- *Data is modified/queried using DML (data manipulation language).*

24

Example: University Database

25

- Conceptual schema:
  - *Students(sid: string, name: string, login: string, age: integer, gpa:real)*
  - *Courses(cid: string, cname:string, credits:integer)*
  - *Enrolled(sid:string, cid:string, grade:string)*
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.
- External Schema (View):
  - *Course_info(cid:string,enrollment:integer)*

25

Integrity Constraints

26

- An *integrity constraint* is a property that must be satisfied by all meaningful database instances.
- A constraint can be seen as *a predicate*; a database is *legal* if it satisfies all integrity constraints.
- Types of constraints
  - Intra-relational constraints: e.g., *domain constraints* and *tuple constraints*
  - Inter-relational constraints: most common is *referential constraint*

26

Tuple and Domain Constraints

27

- A *tuple constraint* expresses conditions on the values of each tuple, independently of other tuples.
- E.g., `Net = Amount-Deductions`
- A *domain constraint* is a tuple constraint that involves a single attribute
- e.g., `(GPA ≤ 4.0) AND (GPA ≥ 0.0)`

27

Unique Values for Tuples

28

RegNum	Surname	FirstName	BirthDate	DegreeProg
284328	Smith	Luigi	29/04/59	Computing
296328	Smith	John	29/04/59	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	01/05/61	Fine Art
965536	Black	Lucy	05/03/58	Fine Art

- Registration number identifies students, i.e., there is no pair of tuples with the same value for `RegNum`.
- Personal data could identify students as well, i.e., there is no pair of tuples with the same values for all of `Surname`, `FirstName`, `BirthDate`.

28

Keys

29

- A *key* is a set of attributes that uniquely identifies tuples in a relation.
- More precisely:
  - A set of attributes K is a *superkey* for a relation r if r cannot contain two distinct tuples t_1 and t_2 such that $t_1[K]=t_2[K]$;
  - K is a (candidate) *key* for r if K is a minimal superkey

  (that is, there exists no other superkey K' of r that is contained in K as proper subset, i.e, $K' \subset K$)

29

Announcements

16

- Use db2srv3 server to access SE3DB3 database instance

16

Unique Values for Tuples

17

RegNum	Surname	FirstName	BirthDate	DegreeProg
284328	Smith	Luigi	29/04/59	Computing
296328	Smith	John	29/04/59	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	01/05/61	Fine Art
965536	Black	Lucy	05/03/58	Fine Art

- Registration number identifies students, i.e., there is no pair of tuples with the same value for **RegNum**.
- Personal data could identify students as well, i.e., there is no pair of tuples with the same values for all of **Surname**, **FirstName**, **BirthDate**.

17

Keys

18

- A *key* is a set of attributes that uniquely identifies tuples in a relation.
- More precisely:
  - A set of attributes K is a *superkey* for a relation r if r cannot contain two distinct tuples t_1 and t_2 such that $t_1[K]=t_2[K]$;
  - K is a (candidate) *key* for r if K is a minimal superkey

  (that is, there exists no other superkey K' of r that is contained in K as proper subset, i.e, $K' \subset K$)

18

Example

19

RegNum	Surname	FirstName	BirthDate	DegreeProg
284328	Smith	Luigi	29/04/59	Computing
296328	Smith	John	29/04/59	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	01/05/61	Fine Art
965536	Black	Lucy	05/03/58	Fine Art

- **RegNum** is a key: i.e., **RegNum** is a superkey and it contains a sole attribute, so it is minimal.
- **{Surname, Firstname, BirthDate}** is another key

19

Beware!

20

RegNum	Surname	FirstName	BirthDate	DegreeProg
296328	Smith	John	29/04/59	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	01/05/61	Fine Art
965536	Black	Lucy	05/03/58	Engineering

- There is no pair of tuples with the same values on both **Surname** and **DegreeProg**;

  i.e., in each program students have different surnames; can we conclude that **Surname** and **DegreeProg** form a key for this relation?

  No! There ***could be*** students with the same surname in the same program

20

Existence of Keys

21

- Relations are sets; therefore each relation is composed of distinct tuples.
- It follows that the whole set of attributes for a relation defines a **superkey**.
- Therefore **each relation has a key**, which is the set of all its attributes, or a subset thereof.
- The existence of keys guarantees that each piece of data in the database can be accessed,
- Keys are a major feature of the Relational Model and allow us to say that it is "**value-based**".

21

Keys and Null Values

22

If there are nulls, keys do not work that well:

- They do not guarantee unique identification;
- They do not help in establishing correspondences between data in different relations

RegNum	Surname	FirstName	BirthDate	DegreeProg
NULL	Smith	John	NULL	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	NULL	NULL
NULL	Black	Lucy	05/03/58	Engineering

- Are the third and fourth tuple the same?
- How do we access the first tuple?

22

Primary Keys

23

- The presence of nulls in keys has to be limited.
- Each relation must have a *primary key* on which nulls are not allowed (in any attribute)
- Notation: the attributes of the primary key are *underlined*
- References between relations are realized through primary keys

RegNum	Surname	FirstName	BirthDate	DegreeProg
643976	Smith	John	NULL	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	NULL	NULL
735591	Black	Lucy	05/03/58	Engineering

23

Do we Always Have Primary Keys?

24

- In most cases, we do have reasonable primary keys (e.g., student number, SIN)
- There may be multiple keys, one of which is designated as primary.

24

Recap

25

- A set of fields is a *key* for a relation if:
  1. No two distinct tuples can have same values in all key fields, and
  2. This is not true for any subset of the key.
- If #2 false, then a *superkey*.
- If there's >1 key for a relation, one of the keys is chosen to be the *primary key*.
- E.g., *sid* is a key for Students. (What about *name*?) The set {*sid, gpa*} is a superkey.

25

Primary and Candidate Keys

26

Enrolled(sid, cid, grade)

1. "For a given student and course, there is a single grade." vs.
2. "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."

Enrolled(sid, cid, grade)

- Be careful to define Integrity Constraints (ICs) correctly at design time.
- ICS are checked when data is updated.

Enrolled(sid, cid, grade)

• key (cid, grade)

26

Foreign Keys

27

- Pieces of data in different relations are correlated by means of values of primary keys.
- Referential integrity constraints are imposed in order to guarantee that the values refer to existing tuples in the referenced relation.
- A *foreign key* requires that the values on a set X of attributes of a relation R_1 must appear as values for the primary key of another relation R_2.
  - In other words, set of attributes in one relation that is used to `refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a `logical pointer'.

27

Referential Integrity

28

- E.g. *sid* is a foreign key referring to Students:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.

28

Referential Integrity (cont'd)

29

- Only students listed in the Students relation should be allowed to enroll for courses.

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

29

Enforcing Referential Integrity

30

- Consider Students and Enrolled; *sid* in Enrolled is a foreign key that references Students.
- What should be done if an Enrolled tuple with a non-existent student id is inserted? *Reject it!*
- What should be done if a Students tuple is deleted?
  - Also delete all Enrolled tuples that refer to it.
  - Disallow deletion of a Students tuple that is referred to.
  - Set sid in Enrolled tuples that refer to it to a *default sid*.
  - Set sid in Enrolled tuples that refer to it to NULL.
- Similar if primary key of Students tuple is updated.

30

Where do ICs Come From?

31

- ICs are based upon the semantics of the real-world enterprise that is being described in the database relations.
- We can check a database instance to see if an IC is violated, but we cannot infer that an IC is true by looking at an instance.
  - An IC is a statement about *all possible* instances
- Key and foreign key ICs are the most common; more general ICs supported too.

31

One More Example

32

Offences

Code	Date	Officer	Dept	Registration
143256	25/10/1992	567	75	5694 FR
987554	26/10/1992	456	75	5694 FR
987557	26/10/1992	456	75	6544 XY
630876	15/10/1992	456	47	6544 XY
539856	12/10/1992	567	47	6544 XY

Officers

RegNum	Surname	FirstName
567	Brun	Jean
456	Larue	Henri
638	Larue	Jacques

Cars

Registration	Dept	Owner
6544 XY	75	Cordon Edouard
7122 HT	75	Cordon Edouard
5694 FR	75	Latour Hortense
6544 XY	47	Mimault Bernard

- Offences[Officer] $\subseteq$ Officers[RegNum]
- Offences[Registration,Dept] $\subseteq$ Cars[Registration,Dept]

32

Violation of Foreign keys

33

Offences

Code	Date	Officer	Dept	Registration
987554	26/10/1992	456	75	5694 FR
630876	15/10/1992	456	47	6544 XY

Officers

RegNum	Surname	FirstName
567	Brun	Jean
638	Larue	Jacques

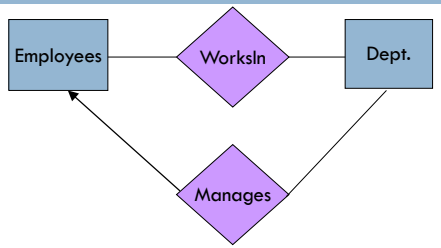
Cars

Registration	Dept	Owner	…
7122 HT	75	Cordon Edouard	…
5694 FR	93	Latour Hortense	…
6544 XY	47	Mimault Bernard	…

33

Key Constraints

37



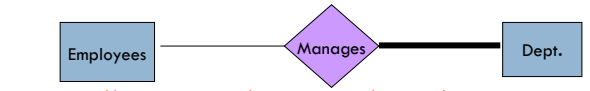
- Many-many: "An employee can work in many depts, and a dept. can have many employees
- One-many: A dept has at most one manager, and employees can manage many departments

37

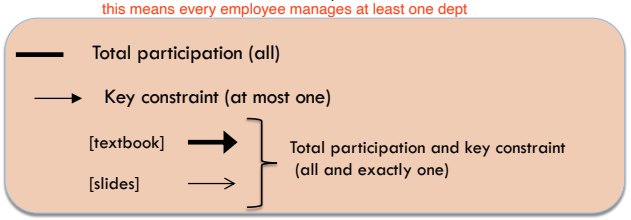
Participation Constraints

38

- Does every dept. have to have a manager?
  - If yes, then every dept. must appear in the manages relation: total participation (vs. partial)



this means every employee manages at least one dept



38

Attributes on Relationships

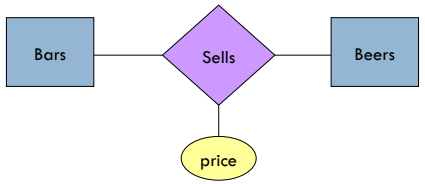
39

- Sometimes it is useful to attach an attribute to a relationship.
- Think of this attribute as a property of tuples in the relationship set.

39

Example: Attribute on Relationship

40



Price is a function of both the bar and the beer, not of one alone.
E.g., "The price of Miller beer at Joe's bar"

40

Equivalent Diagrams Without Attributes on Relationships

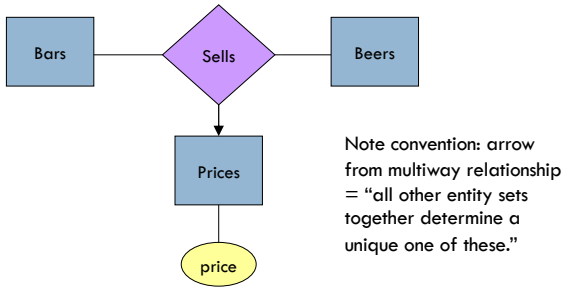
41

- Create an entity set representing values of the attribute.
- Make that entity set participate in the relationship.

41

Example: Removing an Attribute from a Relationship

42



42

Roles

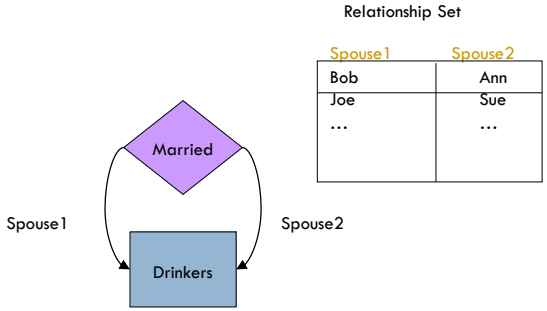
43

- Sometimes an entity set appears more than once in a relationship.
- Label the edges between the relationship and the entity set with names called *roles*.

43

Example: Roles

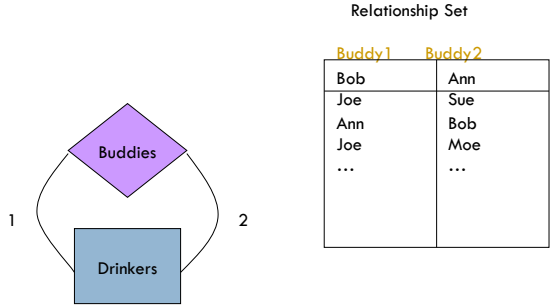
44



44

Example: Roles

45



Relationship Set

Buddy1	Buddy2
Bob	Ann
Joe	Sue
Ann	Bob
Joe	Moe
…	…

45

Subclasses

46

- *Subclass* = special case = more properties.
- Example: Ales are a kind of beer.
  - Not every beer is an ale, but some are.
  - Let us suppose that in addition to all the *properties* (attributes and relationships) of beers, ales also have the attribute color.

46

Subclasses in E/R Diagrams

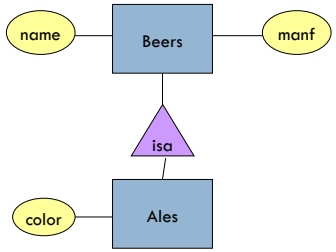
47

- isa triangles indicate the subclass relationship.
  - Point to the superclass.
- Reasons for using isa:
  - To add descriptive attributes specific to a subclass.
  - To identify entities that participate in a relationship.

47

Example: Subclasses

48



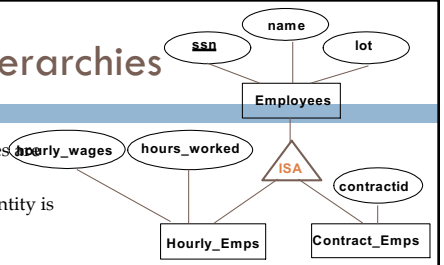
Assume subclasses form a tree.

48

ISA (`is a') Hierarchies

49

- As in C++, or other PLs, attributes are inherited.
- If we declare A **ISA** B, every A entity is also considered to be a B entity.



- *Overlap constraints*: Can two sub-classes contain the same entity? E.g., Can Joe be an Hourly_Emps as well as a Contract_Emps entity?
- *Covering constraints*: Does every Employees entity have to be an Hourly_Emps or a Contract_Emps entity?

R. Ramakrishnan & J. Gehrke

49

Keys

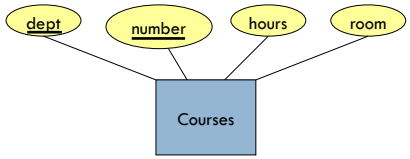
50

- A *key* is a set of attributes for one entity set such that no two entities in this set agree on all the attributes of the key.
  - It is allowed for two entities to agree on some, but not all, of the key attributes.
- We must designate a key for every entity set.
- Underline the key attribute(s).

50

Example: a Multi-attribute Key

51



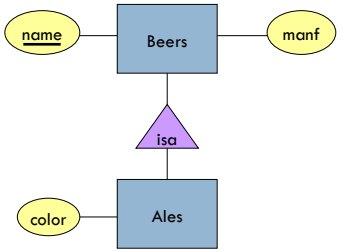
- Note that hours and room could also serve as a key, but we must select only one primary key.

51

Keys

52

In an Isa hierarchy, only the root entity set has a key, and it must serve as the key for all entities in the hierarchy.



52

Weak Entity Sets

53

- Occasionally, entities of an entity set need "help" to identify them uniquely.
- Entity set *E* is said to be *weak* if in order to identify entities of *E* uniquely, we need to follow one or more many-one relationships from *E* and include the key of the related entities from the connected entity sets.

53

Example: Weak Entity Set

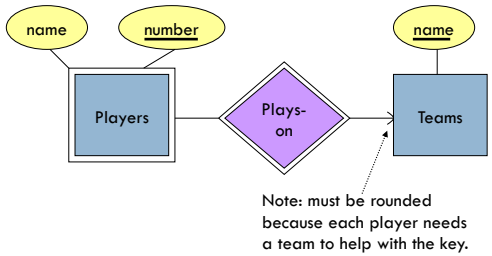
54

- name is almost a key for football players, but there might be two with the same name.
- number is certainly not a key, since players on two teams could have the same number.
- But number, together with the team name related to the player by Plays-on should be unique.

54

In E/R Diagrams

55



- Double diamond for *supporting* many-one relationship.
- Double rectangle for the weak entity set.

55

Weak Entity-Set Rules

56

- A weak entity set has one or more many-one relationships to other (supporting) entity sets.
  - Not every many-one relationship from a weak entity set need be supporting.
  - But supporting relationships must have a rounded arrow (entity at the "one" end is guaranteed).

56

Weak Entity-Set Rules – (2)

57

- The key for a weak entity set is its own underlined attributes and the keys from the supporting entity sets.
  - E.g., (player) number and (team) name is a key for Players in the previous example.

57

Design Techniques

58

1. Avoid redundancy.
2. Limit the use of weak entity sets.
3. Don't use an entity set when an attribute will do.

58

Avoiding Redundancy

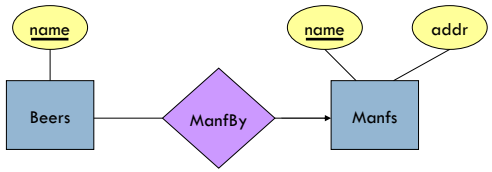
59

- *Redundancy* = saying the same thing in two (or more) different ways.
- Wastes space and (more importantly) encourages inconsistency.
  - Two representations of the same fact become inconsistent if we change one and forget to change the other.

59

Example: Good

60

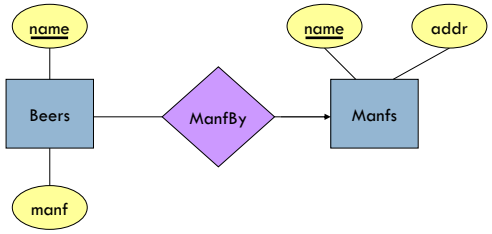


This design gives the address of each manufacturer exactly once.

60

Example: Bad

61

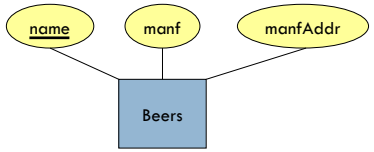


This design states the manufacturer of a beer twice: as an attribute and as a related entity.

61

Example: Bad

62



This design repeats the manufacturer's address once for each beer and loses the address if there are temporarily no beers for a manufacturer.

62

Entity Sets Versus Attributes

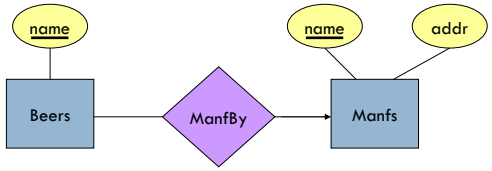
1

- An entity set should satisfy at least one of the following conditions:
  - It is more than the name of something; it has at least one non-key attribute. OR
  - It is the "many" in a many-one or many-many relationship.
- Depends on the application requirements:
  - If we have several addresses per employee, *address* must be an entity (since attributes cannot be set-valued).
  - If the structure (city, street, etc.) is important, e.g., we want to retrieve employees in a given city, *address* must be modeled as an entity (since attribute values are atomic).

1

Example: Good

2

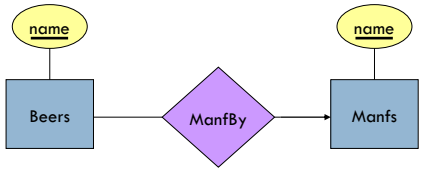


•Manfs deserves to be an entity set because of the nonkey attribute addr.
•Beers deserves to be an entity set because it is the "many" of the many-one relationship ManfBy.

2

Example: Bad

3

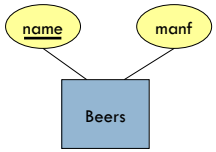


Since the manufacturer is nothing but a name, and is not at the "many" end of any relationship, it need not be an entity set.

3

Example: Good

4



There is no need to make the manufacturer an entity set, because we record nothing about manufacturers besides their name.

4

Don't Overuse Weak Entity Sets

5

- Beginning database designers often doubt that anything could be a key by itself.
  - They make all entity sets weak, supported by all other entity sets to which they are linked.
- In reality, we usually create unique ID's for entity sets.
  - Examples include social-security numbers, automobile VIN's etc.

5

When Do We Need Weak Entity Sets?

6

- The usual reason is that there is no global authority capable of creating unique ID's.
- Example: it is unlikely that there could be an agreement to assign unique player numbers across all football teams in the world.

6

From E/R Diagrams to Relations

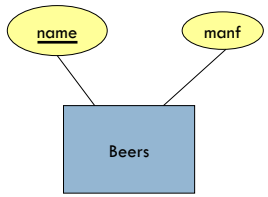
7

- Entity set -> relation.
  - Attributes -> attributes.
- Relationships -> relations whose attributes are only:
  - The keys of the connected entity sets.
  - Attributes of the relationship itself.

7

Entity Set -> Relation

8

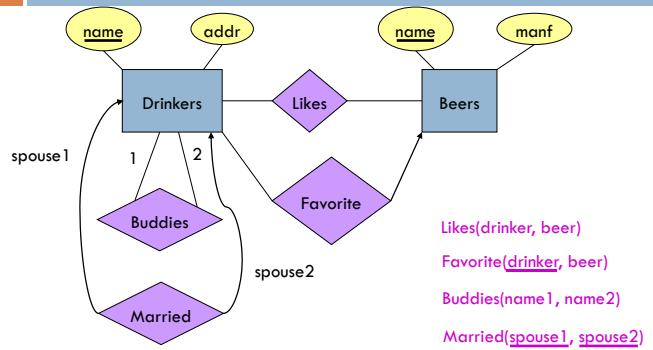


Relation: Beers(name, manf)

8

Relationship -> Relation

9



9

Handling Weak Entity Sets

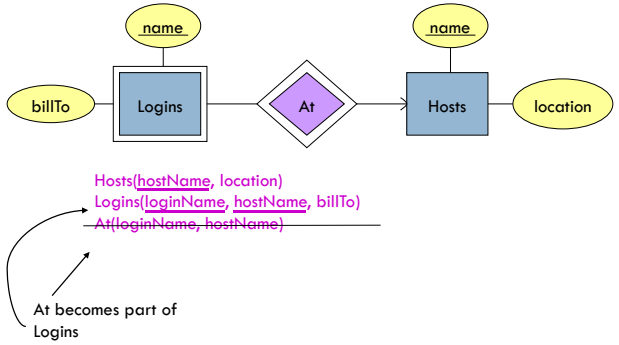
10

- Relation for a weak entity set must include attributes for its complete key (including those belonging to other entity sets), as well as its own, nonkey attributes.
- A supporting relationship is redundant and yields no relation (unless *it* has attributes).

10

Example: Weak Entity Set -> Relation

11

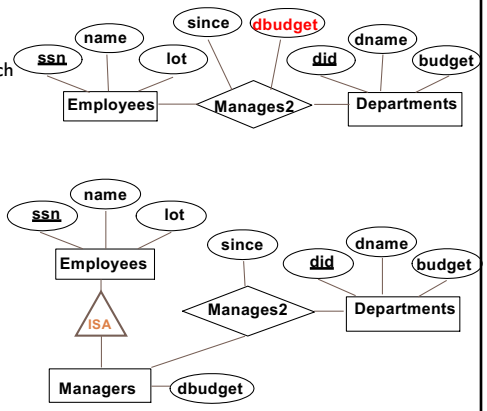


11

Entity vs. Relationship

12

- First ER diagram OK if a manager gets a separate discretionary budget for each dept.
- What if a manager gets a discretionary budget that covers *all* managed depts?
  - Redundancy: *dbudget* stored for each dept managed by manager.
  - Misleading: Suggests *dbudget* associated with department-mgr combination.



12

Summary

13

- *Conceptual design* follows *requirements analysis*,
  - Yields a high-level description of data to be stored
- ER model popular for conceptual design
  - Constructs are expressive, close to the way people think about their applications.
- Basic constructs: *entities*, *relationships*, and *attributes* (of entities and relationships).
- Some additional constructs: *weak entities*, *ISA hierarchies*.

R. Ramakrishnan & J. Gehrke

13

Summary of ER (cont'd.)

14

- Several kinds of integrity constraints can be expressed in the ER model: *key constraints*, *participation constraints*, and *overlap/covering constraints* for ISA hierarchies.
  - Constraints play an important role in determining the best database design for an enterprise.

14

Summary (cont'd)

15

- ER design is *subjective*. There are often many ways to model a given scenario!
- Analyzing alternatives can be tricky, especially for a large enterprise. Common choices include:
  - Entity vs. attribute, entity vs. relationship, binary or n-ary relationship, whether or not to use ISA hierarchies
  - Ensuring good database design: resulting relational schema should be analyzed and refined further.

15

SQL: DATA DEFINITION LANGUAGE

16

Database Schemas in SQL

17

- SQL is primarily a query language, for getting information from a database.
  - **Data manipulation language (DML)**
- But SQL also includes a *data-definition* component for describing database schemas.
  - **Data definition language (DDL)**

17

Creating (Declaring) a Relation

18

- Simplest form is:

```
CREATE TABLE <name> (
        <list of elements>
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

18

Elements of Table Declarations

19

- Most basic element: an attribute and its type.
- The most common types are:
  - INT or INTEGER (synonyms).
  - REAL or FLOAT (synonyms).
  - CHAR(*n*) = fixed-length string of *n* characters.
  - VARCHAR(*n*) = variable-length string of up to *n* characters.

19

Example: Create Table

20

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer    VARCHAR(20),
    price   REAL
);
```

20

SQL Values

21

- Integers and reals are represented as you would expect.
- Strings are too, except they require single quotes.
  - Two single quotes = real quote, e.g., `'Joe''s Bar'`.
- Any value can be NULL
  - Unless attribute has NOT NULL constraint
  - E.g., price REAL not null,

21

Dates and Times

22

- DATE and TIME are types in SQL.
- The form of a date value is:

  DATE 'yyyy-mm-dd'
  - Example: `DATE '2007-09-30'` for Sept. 30, 2007.

22

Times as Values

23

- The form of a time value is:

  TIME 'hh:mm:ss'

  with an optional decimal point and fractions of a second following.
  - Example: `TIME '15:30:02.5'` = two and a half seconds after 3:30PM.

23

Declaring Keys

24

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE.
- Either says that no two tuples of the relation may agree in all the attribute(s) on the list.

24

Our Running Example

25

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

- Underline = *key* (tuples cannot have the same value in all key attributes).

25

Declaring Single-Attribute Keys

26

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE Beers (
    name   CHAR(20) UNIQUE,
    manf   CHAR(20)
);
```

26

Declaring Multiattribute Keys

27

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement.
- This form is essential if the key consists of more than one attribute.
  - May be used even for one-attribute keys.

27

Example: Multiattribute Key

28

- The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
    bar     CHAR(20),
    beer    VARCHAR(20),
    price   REAL,
    PRIMARY KEY (bar, beer)
);
```

28

PRIMARY KEY vs. UNIQUE

29

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

29

Declaring Single-Attribute Keys

1

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute.
- Example:

```
CREATE TABLE Beers (
    name   CHAR(20) UNIQUE,
    manf   CHAR(20)
);
```

1

Declaring Multiattribute Keys

2

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement.
- This form is essential if the key consists of more than one attribute.
  - May be used even for one-attribute keys.

2

Example: Multiattribute Key

3

- The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
    bar      CHAR(20),
    beer     VARCHAR(20),
    price    REAL,
    PRIMARY KEY (bar, beer)
);
```

3

PRIMARY KEY vs. UNIQUE

4

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL.

4

Kinds of Constraints

5

- Keys
- Foreign-key, or referential-integrity.
- Domain constraints
  - Constrain values of a particular attribute.
- Tuple-based constraints
  - Relationship among components.
- Assertions: any SQL boolean expression

5

Foreign Keys

6

- Values appearing in attributes of one relation must appear together in certain attributes of another relation.
- Example: in Sells(bar, beer, price), we might expect that a beer value also appears in Beers.name

6

Expressing Foreign Keys

7

- Use keyword REFERENCES, either:
  1. After an attribute (for one-attribute keys).
  2. As an element of the schema:

  FOREIGN KEY (<list of attributes>)
  REFERENCES <relation> (<attributes>)
- Referenced attributes must be declared PRIMARY KEY or UNIQUE.

7

Example: With Attribute

8

```
CREATE TABLE Beers (
  name     CHAR(20) PRIMARY KEY,
  manf     CHAR(20) );

CREATE TABLE Sells (
  bar      CHAR(20),
  beer     CHAR(20) REFERENCES Beers(name),
  price    REAL );
```

8

Example: As Schema Element

9

```
CREATE TABLE Beers (
  name     CHAR(20) PRIMARY KEY,
  manf     CHAR(20) );

CREATE TABLE Sells (
  bar      CHAR(20),
  beer     CHAR(20),
  price    REAL,
  FOREIGN KEY(beer) REFERENCES
     Beers(name));
```

9

Enforcing Foreign-Key Constraints

10

- If there is a foreign-key constraint from relation *R* to relation *S*, two violations are possible:
  1. An insert or update to *R* introduces values not found in *S*.
  2. A deletion or update to S causes some tuples of *R* to "dangle."

10

Actions Taken --- (1)

11

- Example: suppose *R* = Sells, *S* = Beers.
- An insert or update to Sells that introduces a nonexistent beer must be rejected.
- A deletion or update to Beers that removes a beer value found in some tuples of Sells can be handled in three ways…

11

Actions Taken --- (2)

12

1. *Default* : Reject the modification.
2. *Cascade* : Make the same changes in Sells.
   - Deleted beer: delete Sells tuple.
   - Updated beer: change value in Sells.
3. *Set NULL* : Change the beer to NULL.

12

Example: Cascade

13

- Delete the Bud tuple from Beers:
  - Then delete all tuples from Sells that have beer = 'Bud'.
- Update the Bud tuple by changing 'Bud' to 'Budweiser':
  - Then change all Sells tuples with beer = 'Bud' to beer = 'Budweiser'.

13

Example: Set NULL

14

- Delete the Bud tuple from Beers:
  - Change all tuples of Sells that have beer = 'Bud' to have beer = NULL.
- Update the Bud tuple by changing 'Bud' to 'Budweiser':
  - Same change as for deletion.

14

Choosing a Policy

15

- When we declare a foreign key, we may choose policies SET NULL or CASCADE independently for deletions and updates.
- Follow the foreign-key declaration by:

ON [UPDATE, DELETE][SET NULL CASCADE]

- Two such clauses may be used.
- Otherwise, the default (reject) is used.

15

Example: Setting Policy

16

```
CREATE TABLE Sells (
  bar    CHAR(20),
  beer   CHAR(20),
  price  REAL,
  FOREIGN KEY(beer)
     REFERENCES Beers(name)
     ON DELETE SET NULL
     ON UPDATE CASCADE
);
```

16

Attribute-Based Checks

17

- Constraints on the value of a particular attribute.
- Add CHECK(<condition>) to the declaration for the attribute.
- The condition may use the name of the attribute, but any other relation or attribute name must be in a subquery.

17

Example: Attribute-Based Check

18

```
CREATE TABLE Sells (
  bar    CHAR(20),
  beer   CHAR(20) CHECK ( beer IN
         (SELECT name FROM Beers)),
  price  REAL CHECK ( price <= 5.00 )
);
```

18

Timing of Checks

19

- Attribute-based checks are performed only when a value for that attribute is inserted or updated.
  - Example: `CHECK (price <= 5.00)` checks every new price and rejects the modification (for that tuple) if the price is more than $5.
  - Example: `CHECK (beer IN (SELECT name FROM Beers))` not checked if a beer is deleted from Beers (unlike foreign-keys).

19

Tuple-Based Checks

20

- CHECK (<condition>) may be added as a relation-schema element.
- The condition may refer to any attribute of the relation.
  - But other attributes or relations require a subquery.
- Checked on insert or update only.

20

Example: Tuple-Based Check

21

- Only Joe's Bar can sell beer for more than $5:

```
CREATE TABLE Sells (
    bar       CHAR(20),
    beer      CHAR(20),
    price     REAL,
    CHECK (bar = 'Joe''s Bar' OR
                  price <= 5.00)
);
```

21

INTRODUCTION TO SQL

22

Why SQL?

23

- SQL is a very-high-level language.
  - Structured Query Language
  - Say "what to do" rather than "how to do it."
  - Avoid a lot of data-manipulation details needed in procedural languages
  - Database management system figures out "best" way to execute query.
  - Called "query optimization."

Credit: Renee J. Miller

23

Database Schemas in SQL

24

- SQL is primarily a query language, for getting information from a database.
  - **Data manipulation language (DML)**
- But SQL also includes a *data-definition* component for describing database schemas.
  - **Data definition language (DDL)**

24

Select-From-Where Statements

25

SELECT desired attributes

FROM one or more tables

WHERE condition about tuples of the tables

25

Our Running Example

26

- Our SQL queries will be based on the following database schema.
  - Underline indicates key attributes.

Beers(name, manf)
Bars(name, addr, license)
Drinkers(name, addr, phone)
Likes(drinker, beer)
Sells(bar, beer, price)
Frequents(drinker, bar)

26

Example

27

- Using Beers(name, manf), what beers are made by Anheuser-Busch?

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

27

Result of Query

28

name
Bud
Bud Lite
Michelob
. . .

The answer is a relation with a single attribute, name, and tuples with the name of each beer by Anheuser-Busch, such as Bud.

28

Meaning of Single-Relation Query

29

- Begin with the relation in the FROM clause.
- Apply the selection indicated by the WHERE clause.
- Apply the extended projection indicated by the SELECT clause.

29

Operational Semantics - General

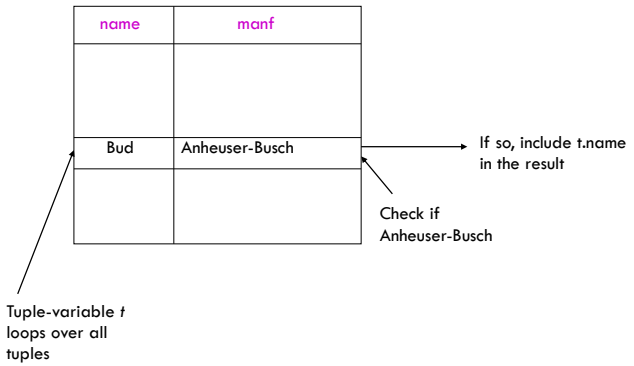
30

- Think of a *tuple variable* visiting each tuple of the relation mentioned in FROM.
- Check if the tuple assigned to the tuple variable satisfies the WHERE clause.
- If so, compute the attributes or expressions of the SELECT clause using the components of this tuple.

30

Operational Semantics

31



31

Example

32

- What beers are made by Anheuser-Busch?

```
SELECT name
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

OR:

```
SELECT t.name
FROM Beers t
WHERE t.manf ='Anheuser-Busch';
```

Note: these are identical queries.

32

* In SELECT clauses

33

- When there is one relation in the FROM clause, * in the SELECT clause stands for "all attributes of this relation."
- Example: Using Beers(name, manf):

```
SELECT *
FROM Beers
WHERE manf = 'Anheuser-Busch';
```

33

Result of Query:

34

name	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
...	...

Now, the result has each of the attributes of Beers.

34

Renaming Attributes

35

- If you want the result to have different attribute names, use "AS <new name>" to rename an attribute.
- Example: Using Beers(name, manf):

```
SELECT name AS beer, manf
FROM Beers
WHERE manf = 'Anheuser-Busch'
```

35

Result of Query:

36

beer	manf
Bud	Anheuser-Busch
Bud Lite	Anheuser-Busch
Michelob	Anheuser-Busch
...	...

36

Expressions in SELECT Clauses

37

- Any valid expression can appear as an element of a SELECT clause.
- Example: Using Sells(bar, beer, price):

```
SELECT bar, beer,
       price*95 AS priceInYen
FROM Sells;
```

37

Result of Query

38

bar	beer	priceInYen
Joe's	Bud	285
Sue's	Miller	342
…	…	…

38

Example: Constants as Expressions

39

- Using Likes(drinker, beer):

```
SELECT drinker,
       'likes Bud' AS whoLikesBud
FROM Likes
WHERE beer = 'Bud';
```

39

Result of Query

40

drinker	whoLikesBud
Sally	likes Bud
Fred	likes Bud
…	…

40

Complex Conditions in WHERE Clause

41

- Boolean operators AND, OR, NOT.
- Comparisons =, <>, <, >, <=, >=.

41

Example: Complex Condition

42

- Using Sells(bar, beer, price), find the price Joe's Bar charges for Bud:

```
SELECT  price
FROM    Sells
WHERE bar = 'Joe''s Bar' AND
      beer = 'Bud';
```

42

Patterns

22

- A condition can compare a string to a pattern by:
  - <Attribute> LIKE <pattern> or <Attribute> NOT LIKE <pattern>
- *Pattern* is a quoted string
  - % = "any string";
  - _ = "any character".

22

Example: LIKE

23

- Using Drinkers(name, addr, phone) find the drinkers with exchange 555:

```
SELECT name
FROM Drinkers
WHERE phone LIKE '%555-_ _ _ _';
```

23

NULL Values

24

- Tuples in SQL relations can have NULL as a value for one or more components.
- Meaning depends on context. Two common cases:
  - *Missing value* : e.g., we know Joe's Bar has some address, but we don't know what it is.
  - *Inapplicable* : e.g., the value of attribute spouse for an unmarried person.

24

Comparing NULL's to Values

25

- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN.
- Comparing any value (including NULL itself) with NULL yields UNKNOWN.
- A tuple is in a query answer iff the WHERE clause is TRUE (not FALSE or UNKNOWN).

25

Three-Valued Logic

26

- To understand how AND, OR, and NOT work in 3-valued logic
- For TRUE result
  - OR: at least one operand must be TRUE
  - AND: both operands must be TRUE
  - NOT: operand must be FALSE
- For FALSE result
  - OR: both operands must be FALSE
  - AND: at least one operand must be FALSE
  - NOT: operand must be TRUE
- Otherwise, result is UNKNOWN

26

Example

27

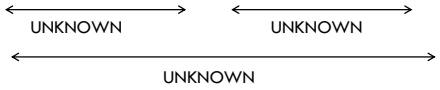
- From the following Sells relation:

bar	beer	price
Joe's Bar	Bud	NULL

SELECT bar

FROM Sells

WHERE price < 2.00 OR price >= 5.00;



27

Multi-Relation Queries

28

- Interesting queries often combine data from more than one relation.
- We can address several relations in one query by listing them all in the FROM clause.
- Distinguish attributes of the same name by "<relation>.<attribute>" .

28

Example: Joining Two Relations

29

- Using relations Likes(drinker, beer) and Frequents(drinker, bar), find the beers liked by at least one person who frequents Joe's Bar.

```
SELECT beer
FROM Likes, Frequents
WHERE bar = 'Joe''s Bar' AND
      Frequents.drinker = Likes.drinker;
```

29

Example: Joining Two Relations

30

- Alternatively can use explicit (named) tuple variables

```
SELECT beer
FROM Likes l, Frequents f
WHERE bar = 'Joe''s Bar' AND
   f.drinker = l.drinker;
```

30

Formal Semantics

31

- Almost the same as for single-relation queries:
  - Start with the product of all the relations in the FROM clause.
  - Apply the selection condition from the WHERE clause.
  - Project onto the list of attributes and expressions in the SELECT clause.

31

Operational Semantics

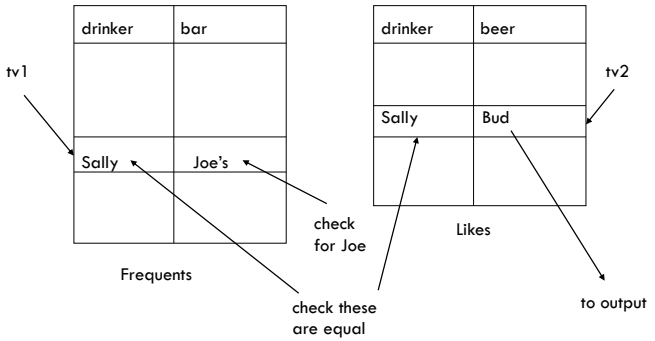
32

- Imagine one tuple-variable for each relation in the FROM clause.
  - These tuple-variables visit each combination of tuples, one from each relation.
- If the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause.

32

Example

33



33

Explicit Tuple-Variables

34

- Sometimes, a query needs to use two copies of the same relation.
- Distinguish copies by following the relation name by the name of a tuple-variable, in the FROM clause.
- It's always an option to rename relations this way, even when not essential.

34

Example: Self-Join

35

- From Beers(name, manf), find all pairs of beers by the same manufacturer.
  - Do not produce pairs like (Bud, Bud).
  - Do not produce the same pair twice like (Bud, Miller) and (Miller, Bud).

```
SELECT b1.name, b2.name
FROM Beers b1, Beers b2
WHERE b1.manf = b2.manf AND
      b1.name < b2.name;
```

35

Subqueries

36

- A parenthesized SELECT-FROM-WHERE statement (*subquery*) can be used as a value in a number of places, including FROM and WHERE clauses.
- Example: in place of a relation in the FROM clause, we can use a subquery and then query its result.
  - Must use a tuple-variable to name tuples of the result.

36

Example: Subquery in FROM

37

- Find the beers liked by at least one person who frequents Joe's Bar.

Drinkers who frequent Joe's Bar

```
SELECT beer
FROM Likes, (SELECT drinker
             FROM Frequents
             WHERE bar = 'Joe''s Bar') JD
WHERE Likes.drinker = JD.drinker;
```

37

Subqueries often obscure queries

38

- Find the beers liked by at least one person who frequents Joe's Bar.

```
SELECT beer
FROM Likes l, Frequents f
WHERE l.drinker = f.drinker AND
   bar = 'Joe''s Bar';
```

Simple join query

38

Subqueries That Return One Tuple

39

- If a subquery is guaranteed to produce one tuple, then the subquery can be used as a value.
  - Usually, the tuple has one component.
  - Remember SQL's 3-valued logic.

39

Example: Single-Tuple Subquery

40

- Using Sells(bar, beer, price), find the bars that serve Miller for the same price Joe charges for Bud.

Two queries would work:

- Find the price Joe charges for Bud.
- Find the bars that serve Miller at that price.

40

Query + Subquery Solution

41

- Find the price Joe charges for Bud.
- Find the bars that serve Miller at that price.

Sells(bar, beer, price)

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND price
   = (SELECT price
      FROM Sells
      WHERE bar = 'Joe''s Bar'
         AND beer = 'Bud');
```

The price at which Joe sells Bud

What if price of Bud is NULL?

41

Query + Subquery Solution

42

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
      price = (SELECT price
               FROM Sells
               WHERE beer = 'Bud');
```

What if subquery returns multiple values?

42

Query + Subquery Solution

41

SELECT bar
FROM Sells
WHERE beer = 'Miller' AND price
= (SELECT price
FROM Sells
WHERE bar = 'Joe''s Bar'
AND beer = 'Bud');

- Find the price Joe charges for Bud.
- Find the bars that serve Miller at that price.

Sells(bar, beer, price)

The price at which Joe sells Bud

What if price of Bud is NULL?

41

Query + Subquery Solution

42

SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
price = (SELECT price
FROM Sells
WHERE beer = 'Bud');

What if subquery returns multiple values?

42

Recap: Conditions in WHERE Clause

43

- Boolean operators AND, OR, NOT.
- Comparisons =, <>, <, >, <=, >=.
- LIKE operator

- SQL includes a **between** comparison operator
- Example: Find the names of all instructors with salary between $90,000 and $100,000 (that is, ≥ $90,000 and ≤ $100,000)
  - **select** *name*
    from *instructor*
    where *salary* **between** 90000 **and** 100000

43

The Operator ANY

44

- *x* = ANY(<subquery>) is a boolean condition that is true iff *x* equals at least one tuple in the subquery result.
  - = could be any comparison operator.
- Example: *x* >= ANY(<subquery>) means *x* is not the uniquely smallest tuple produced by the subquery.
  - Note tuples must have one component only.

44

The Operator ALL

45

- *x* <> ALL(<subquery>) is true iff for every tuple *t* in the relation, *x* is not equal to *t*.
  - That is, *x* is not in the subquery result.
- <> can be any comparison operator.
- Example: *x* >= ALL(<subquery>) means there is no tuple larger than *x* in the subquery result.

45

Example: ALL

46

- From Sells(bar, beer, price), find the beer(s) sold for the highest price.

```
SELECT beer
FROM Sells
WHERE price >=
    ALL( SELECT price
         FROM Sells);
```

price from the outer Sells must not be less than any price.

46

The IN Operator

47

- <value> IN (<subquery>) is true if and only if the <value> is a member of the relation produced by the subquery.
  - Opposite: <value> NOT IN (<subquery>).
- IN-expressions can appear in WHERE clauses.
- WHERE col IN (value1, value2, …)

47

IN is Concise

48

- SELECT * FROM Cartoons
  WHERE LastName IN ('Jetsons', 'Smurfs', 'Flintstones')

- SELECT * FROM Cartoons
  WHERE LastName = 'Jetsons'
  OR LastName = 'Smurfs'
  OR LastName = 'Flintstones'

48

Example: IN

49

- Using Beers(name, manf) and Likes(drinker, beer), find the name and manufacturer of each beer that Fred likes.

```
SELECT *
FROM Beers
WHERE name IN (SELECT beer
               FROM Likes
               WHERE drinker = 'Fred');
```

The set of beers Fred likes

49

IN vs. Join

50

```
SELECT R.a
FROM R, S
WHERE R.b = S.b;

SELECT R.a
FROM R
WHERE b IN (SELECT b FROM S);
```

50

IN is a Predicate About R's Tuples

51

```
SELECT a
FROM R
WHERE b IN (SELECT b FROM S);
```

Two 2's

One loop, over the tuples of R

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) satisfies the condition; 1 is output once.

51

This Query Pairs Tuples from R, S

52

```
SELECT a
FROM R, S
WHERE R.b = S.b;
```

Double loop, over the tuples of R and S

a	b
1	2
3	4

R

b	c
2	5
2	6

S

(1,2) with (2,5) and (1,2) with (2,6) both satisfy the condition; 1 is output twice.

52

Back to our original query…

53

```
SELECT bar
FROM Sells
WHERE beer = 'Miller' AND
    price = (SELECT price
             FROM Sells
             WHERE beer = 'Bud');
```

Use IN() or = ANY()

53

Recap

54

- IN( ) is equivalent to = ANY( )
- For ANY( ), you can use other comparison operators such as >, <,... etc, but not applicable for IN( )
- The < >ANY operator, however, differs from NOT IN:
  - < >ANY means not = a, or not = b, or not = c
  - NOT IN means not = a, and not = b, and not = c.
  - <>ALL means the same as NOT IN.

54

Example: =ANY

55

Sells

Bar	Beer	Price
Jane	Miller	3.00
Joe	Miller	4.00
Joe	Bud	3.00
Jack	Bud	4.00
Tom	Miller	4.50

```
SELECT Bar
FROM Sells
WHERE Beer = 'Miller' AND Price =
        ANY(SELECT Price
            FROM Sells
            WHERE Beer='Bud')
```

Result

Bar
Jane
Joe

55

Example: =ANY

1

Sells

Bar	Beer	Price
Jane	Miller	3.00
Joe	Miller	4.00
Joe	Bud	3.00
Jack	Bud	4.00
Tom	Miller	4.50

```
SELECT Bar
FROM Sells
WHERE Beer = 'Miller' AND Price =
    ANY(SELECT Price
            FROM Sells
            WHERE Beer='Bud')
```

Result

Bar
Jane
Joe

1

The Exists Operator

2

- EXISTS(<subquery>) is true if and only if the subquery result is not empty.
- Example: From Beers(name, manf) , find those beers that are the unique (only) beer made by their manufacturer.

Credit: Renee J. Miller

2

Example: EXISTS

3

```
SELECT name
  FROM Beers b1
  WHERE NOT EXISTS (
      SELECT *
      FROM Beers
      WHERE manf = b1.manf AND
            name <> b1.name);
```

Notice scope rule: manf refers to closest nested FROM with a relation having that attribute. (Some DBMS consider this ambiguous.)

Set of beers with the same manf as b1, but not the same beer

Notice the SQL "not equals" operator

3

Union, Intersection, and Difference

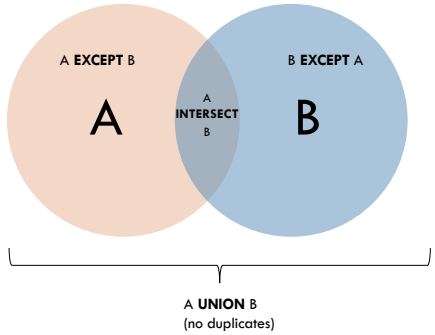
4

- Union, intersection, and difference of relations are expressed by the following forms, each involving subqueries:
  - (<subquery>) UNION (<subquery>)
  - (<subquery>) INTERSECT (<subquery>)
  - (<subquery>) EXCEPT (<subquery>)

4

Visually

5



5

Example: Intersection

6

- Using Likes(drinker, beer), Sells(bar, beer, price), and Frequents(drinker, bar), find the drinkers and beers such that:
  - The drinker likes the beer, and
  - The drinker frequents at least one bar that sells the beer.

6

Solution

7

subquery is really a stored table.

The drinker frequents a bar that sells the beer.

```
(SELECT * FROM Likes)
   INTERSECT
(SELECT drinker, beer
 FROM Sells, Frequents
 WHERE Frequents.bar = Sells.bar
);
```

7

Bag Semantics

8

- A *bag* (or *multiset*) is like a set, but an element may appear more than once.
- Example: {1,2,1,3} is a bag.
- Example: {1,2,3} is also a bag that happens to be a set.

8

Bag (Multiset) Semantics

9

- SQL primarily uses bag semantics
- The SELECT-FROM-WHERE statement uses bag semantics
  - originally for efficiency reasons
- The default for union, intersection, and difference is set semantics.
  - That is, duplicates are eliminated as the operation is applied.

9

Motivation: Efficiency

10

- When doing projection, it is easier to avoid eliminating duplicates.
  - Just work tuple-at-a-time.
- For intersection or difference, it is most efficient to sort the relations first.
  - At that point you may as well eliminate the duplicates anyway.

10

Controlling Duplicate Elimination

11

- Force the result to be a set by SELECT DISTINCT . . .
- Force the result to be a bag (i.e., don't eliminate duplicates) by ALL, as in
  . . . UNION ALL . . .

11

Example: DISTINCT

12

- From Sells(bar, beer, price), find all the different prices charged for beers:

```
SELECT DISTINCT price
FROM Sells;
```

Notice that without DISTINCT, each price would be listed as many times as there were bar/beer pairs at that price.

12

Example: ALL

13

- Using relations Frequents(drinker, bar) and Likes(drinker, beer):
- Lists drinkers who frequent more bars than they like beers, and do so as many times as the difference of those counts.

```
(SELECT drinker FROM Frequents)
   EXCEPT ALL
(SELECT drinker FROM Likes);
```

13

Ordering the Display of Tuples

14

- List in alphabetic order the names of all instructors

  select *name*
  from *instructor*
  order by *name*

- We may specify **desc** for descending order or **asc** for ascending order, for each attribute; ascending order is the default.
  - Example: **order by** *name* **desc**

Credit: Silberchatz, Korth & Sudarshan

14

Humour

15

SQL query walks into a bar, and approaches two tables and asks, can I join you?

15

DATABASE MODIFICATIONS

16

Database Modifications

17

- A *modification* command does not return a result (as a query does), but changes the database in some way.
- Three kinds of modifications:
  1. *Insert* a tuple or tuples.
  2. *Delete* a tuple or tuples.
  3. *Update* the value(s) of an existing tuple or tuples.

17

Insertion

18

- To insert a single tuple:

  INSERT INTO <relation>

  VALUES (<list of values>);
- Example: add to Likes(drinker, beer) the fact that Sally likes Bud.

```
INSERT INTO Likes
VALUES('Sally', 'Bud');
```

18

Specifying Attributes in INSERT

19

- We may add to the relation name a list of attributes.
- Two reasons to do so:
  1. We forget the standard order of attributes for the relation.
  2. We don't have values for all attributes, and we want the system to fill in missing components with NULL or a default value.

19

Example: Specifying Attributes

20

- Another way to add the fact that Sally likes Bud to Likes(drinker, beer):

```
INSERT INTO Likes(beer, drinker)
VALUES('Bud', 'Sally');
```

20

Adding Default Values

21

- In a CREATE TABLE statement, we can follow an attribute by DEFAULT and a value.
- When an inserted tuple has no value for that attribute, the default will be used.

21

Example: Default Values

22

```
CREATE TABLE Drinkers (
   name CHAR(30) PRIMARY KEY,
   addr CHAR(50)
        DEFAULT '123 Sesame St.',
   phone CHAR(16)
);
```

22

Example: Default Values

23

```
INSERT INTO Drinkers(name)
VALUES('Sally');
```

Resulting tuple:

name	address	phone
Sally	123 Sesame St	NULL

23

Inserting Many Tuples

24

- We may insert the entire result of a query into a relation, using the form:

  INSERT INTO <relation>

  (<subquery>);

24

Example: Insert a Subquery

25

- Using Frequents(drinker, bar), enter into the new relation Buddies(name) all of Sally's "potential buddies,"
- i.e., those drinkers who frequent at least one bar that Sally also frequents.

```
INSERT INTO Buddies
(SELECT

);
```

25

Solution

26

"Those drinkers who frequent at least one bar that Sally also frequents"

The other drinker

```
INSERT INTO Buddies
(SELECT d2.drinker
FROM Frequents d1, Frequents d2
WHERE d1.drinker = 'Sally' AND
  d2.drinker <> 'Sally' AND
  d1.bar = d2.bar
);
```

Pairs of Drinker tuples where the first is for Sally, the second is for someone else, and the bars are the same.

26

Deletion

27

- To delete tuples satisfying a condition from some relation:

  DELETE FROM <relation>

  WHERE <condition>;

27

Example: Deletion

28

- Delete from Likes(drinker, beer) the fact that Sally likes Bud:

```
DELETE FROM Likes
WHERE drinker = 'Sally' AND
    beer = 'Bud';
```

28

Example: Delete all Tuples

29

- Make the relation Likes empty:

```
DELETE FROM Likes;
```

- Note no WHERE clause needed.

29

Example: Delete Some Tuples

30

- Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

DELETE FROM Beers b
WHERE

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

30

Example: Delete Some Tuples

1

- Delete from Beers(name, manf) all beers for which there is another beer by the same manufacturer.

```
DELETE FROM Beers b
WHERE
    EXISTS (
        SELECT name
        FROM Beers
        WHERE manf = b.manf AND
                name <> b.name);
```

Beers with the same manufacturer and a different name from the name of the beer represented by tuple b.

1

Semantics of Deletion --- (1)

2

- Suppose Anheuser-Busch makes only Bud and Bud Lite.
- Suppose we come to the tuple *b* for Bud first.
- The subquery is nonempty, because of the Bud Lite tuple, so we delete Bud.
- Now, when *b* is the tuple for Bud Lite, do we delete that tuple too?

2

Semantics of Deletion --- (2)

3

- Answer: we *do* delete Bud Lite as well.
- The reason is that deletion proceeds in two stages:
  1. Mark all tuples for which the WHERE condition is satisfied.
  2. Delete the marked tuples.

3

Updates

4

- To change certain attributes in certain tuples of a relation:

```
UPDATE <relation>
SET <list of attribute assignments>
WHERE <condition on tuples>;
```

4

Example: Update

5

- Change drinker Fred's phone number to 555-1212:

```
UPDATE Drinkers
SET phone = '555-1212'
WHERE name = 'Fred';
```

5

Example: Update Several Tuples

6

- Make $4 the maximum price for beer:

```
UPDATE Sells
SET price = 4.00
WHERE price > 4.00;
```

6

AGGREGATION, GROUPING & OUTER JOINS

7

Aggregation

8

- SUM, AVG, COUNT, MIN, and MAX can be applied to a column in a SELECT clause to produce that aggregation on the column.
- COUNT(*) counts the number of tuples.

8

Example: Aggregation

9

- From Sells(bar, beer, price), find the average price of Bud:

```
SELECT AVG(price)
FROM Sells
WHERE beer = 'Bud';
```

9

Eliminating Duplicates in an Aggregation

10

- Use DISTINCT inside an aggregation.
- Example: find the number of *different* prices charged for Bud:

```
SELECT COUNT(DISTINCT price)
 FROM Sells
 WHERE beer = 'Bud';
```

10

NULL's Ignored in Aggregation

11

- NULL never contributes to a sum, average, or count, and can never be the minimum or maximum of a column.
- But if all the values in a column are NULL, then the result of the aggregation is NULL.
  - Exception: COUNT of an empty set is 0.

11

Example: Effect of NULL's

12

Sells(bar, beer, price)

```
SELECT count(*)
FROM Sells
WHERE beer = 'Bud';
```

The number of bars that sell Bud.

```
SELECT count(price)
FROM Sells
WHERE beer = 'Bud';
```

The number of bars that sell Bud at a known price (i.e., where price is not NULL)

12

Example Query

13

- Find the age of the youngest employee at each rating level

```
SELECT MIN (age)
FROM Employees
WHERE  rating = i
```

13

Grouping

14

- We may follow a SELECT-FROM-WHERE expression by GROUP BY and a list of attributes.
- The relation that results from the SELECT-FROM-WHERE is grouped according to the values of all those attributes, and any aggregation is applied only within each group.

```
SELECT   rating, MIN(age)
FROM     Employees
GROUP BY  rating
```

14

Example: Grouping

15

- From Sells(bar, beer, price), find the average price for each beer:

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer;
```

beer	AVG(price)
Bud	2.33
Miller	4.55
…	…

15

Example: Grouping

16

- From Sells(bar, beer, price) and Frequents(drinker, bar), find for each drinker the average price of Bud at the bars they frequent:

```
SELECT drinker, AVG(price)
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
GROUP BY drinker;
```

Compute all drinker-bar-price triples for Bud.

Then group them by drinker.

16

Restriction on SELECT Lists With Aggregation

17

- If any aggregation is used, then each element of the SELECT list must be either:
  1. Aggregated, or
  2. An attribute on the GROUP BY list.

17

Illegal Query Example

18

```
SELECT bar, beer, MIN(price)
FROM Sells
GROUP BY bar
```

- But this query is illegal in SQL.
- Only one tuple output for each bar, no unique way to select which beer to output

18

A Closer Look

19

```
SELECT bar, beer, MIN(price) AS minP
FROM Sells
GROUP BY bar
```
beer

Sells

Bar	Beer	Price
Joe	Bud	3.00
Joe	Miller	4.00
Tom	Bud	3.50
Tom	Miller	4.25
Jane	Bud	3.25
Jane	Miller	4.75
Jane	Coors	4.00

Result

bar	beer	minP
Joe	?	3.00
Tom	?	3.50
Jane	?	3.25

{Bud, Miller, Coors}?

Only one tuple output for each bar, no unique way to select which beer to output

19

HAVING Clauses

20

- HAVING <condition> may follow a GROUP BY clause.
- If so, the condition applies to each group, and groups not satisfying the condition are eliminated.

20

Example: HAVING

21

- From Sells(bar, beer, price) and Beers(name, manf), find the average price of those beers that are either served in at least three bars or are manufactured by Pete's.

21

Solution

Sells(bar, beer, price) and Beers(name, manf),

22

```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3 OR
    beer IN (SELECT name
             FROM Beers
             WHERE manf = 'Pete''s');
```

Beer groups with at least 3 non-NULL bars

Beers manufactured by Pete's.

22

Requirements on HAVING Conditions

23

- Anything goes in a subquery.
- Outside subqueries, they may refer to attributes only if they are either:
  1. A grouping attribute, or
  2. Aggregated

  (same condition as for SELECT clauses with aggregation).

23

A Final Example

24

```
SELECT Bar, SUM(Qty) AS sumQ
FROM Sells
GROUP BY Bar
HAVING sum(Qty) > 4
```

Sells

Bar	Beer	Price	Qty
Joe	Bud	3.00	2
Joe	Miller	4.00	2
Tom	Bud	3.50	1
Tom	Miller	4.25	4
Jane	Bud	3.25	1
Jane	Miller	4.75	3
Jane	Coors	4.00	2

Result

Bar	sumQ
Tom	5
Jane	6

24

Cross Product

1

- Evaluating joins involves combining two or more relations
- Given two relations, S and R, each row of S is paired with each row of R
- Result schema: one attribute from each attribute of S and R

1

Example

2

Sells

Bar	Beer	Price
Joe	Bud	3.00
Tom	Miller	4.00
Jane	Lite	3.25

Frequents

Drinker	Bar
Aaron	Joe
Mary	Jane

Cross product, also known as the Cartesian product

Sells x Frequents

(Bar)	Beer	Price	Drinker	(Bar)
Joe	Bud	3.00	Aaron	Joe
Joe	Bud	3.00	Mary	Jane
Tom	Miller	4.00	Aaron	Joe
Tom	Miller	4.00	Mary	Jane
Jane	Lite	3.25	Aaron	Joe
Jane	Lite	3.25	Mary	Jane

```
SELECT drinker
FROM Frequents, Sells
WHERE beer = 'Bud' AND
      Frequents.bar = Sells.bar
```

Drinker
Aaron

2

Joined Relations

3

- **Join operations** take two relations and return as a result another relation.
- A join operation is a Cartesian product which requires that tuples in the two relations match (under some condition). It also specifies the attributes that are present in the result of the join

3

Join Operations – Example

4

- Relation *course*

course_id	*title*	*dept_name*	*credits*
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- Relation *prereq*

course_id	*prereq_id*
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- Observe that

  prereq information is missing for CS-315 and

  course information is missing for CS-347

4

Outer Join

5

- An extension of the join operation that avoids loss of information.
- Suppose you have two relations R and S. A tuple of *R* that has no tuple of S with which it joins is said to be *dangling*.
  - Similarly for a tuple of S.
- Computes the join and then adds tuples from one relation that does not match tuples in the other relation to the result of the join.
- Outerjoin preserves dangling tuples by padding them with NULL.

5

Left Outer Join

6

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course **left outer join** prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*

6

Right Outer Join

7

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course **right outer join** prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-347	*null*	*null*	*null*	CS-101

7

Full Outer Join

8

course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

course **full outer join** prereq

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101
CS-315	Robotics	Comp. Sci.	3	*null*
CS-347	*null*	*null*	*null*	CS-101

8

Inner Join

9

- course

course_id	title	dept_name	credits
BIO-301	Genetics	Biology	4
CS-190	Game Design	Comp. Sci.	4
CS-315	Robotics	Comp. Sci.	3

- prereq

course_id	prereq_id
BIO-301	BIO-101
CS-190	CS-101
CS-347	CS-101

- *course* **inner join** *prereq* **on** *course.course_id = prereq.course_id*

course_id	title	dept_name	credits	prereq_id
BIO-301	Genetics	Biology	4	BIO-101
CS-190	Game Design	Comp. Sci.	4	CS-101

9

Outerjoins

10

- R OUTER JOIN S is the core of an outerjoin expression. It is modified by:
  1. Optional NATURAL in front of OUTER.
     - Check equality on all common attributes
     - No two attributes with the same name in the output
  2. Optional ON <condition> after JOIN.
  3. Optional LEFT, RIGHT, or FULL before OUTER.
     - LEFT = pad dangling tuples of R only.
     - RIGHT = pad dangling tuples of S only.
     - FULL = pad both; this choice is the default.

Credit: Renee J. Miller

10

Class Example

11

11

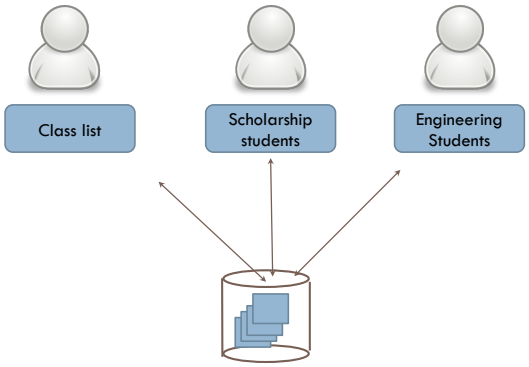
VIEWS

12

12

Scenario

13



13

Views

14

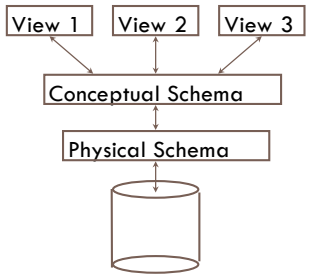
- In most cases, it is not desirable for all users to see the entire data instance.
- A **view** provides a mechanism to hide certain data from the view of certain users.

14

Levels of Abstraction

15

- Many *views*, single *conceptual (logical) schema* and *physical schema*.
  - Views describe how users see the data.
  - Conceptual schema defines logical structure
  - Physical schema describes the files and indexes used.



Credit: Renee J. Miller

15

Views

16

- A *view* is a relation defined in terms of stored tables (called *base tables*) and other views.
- Two kinds:
  1. *Virtual* = not stored in the database; just a query for constructing the relation.
  2. *Materialized* = actually constructed and stored.

16

Declaring Views

17

- Declare by:

  CREATE [MATERIALIZED] VIEW <name> AS <query>;

- A view name
- A possible list of attribute names (for example, when arithmetic operations are specified or when we want the names to be different from the attributes in the base relations)
- A query to specify the view contents
- Default is virtual.

17

Example: View Definition

18

- CanDrink(drinker, beer) is a view "containing" the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer:

```
CREATE VIEW CanDrink AS
   SELECT drinker, beer
   FROM Frequents, Sells
   WHERE Frequents.bar = Sells.bar;
```

18

Example: Accessing a View

19

- Query a view as if it were a base table.
  - Also: a limited ability to modify views if it makes sense as a modification of one underlying base table.
- Example query:

```
SELECT beer FROM CanDrink
WHERE drinker = 'Sally';
```

19

Another Example

20

- Example: View Synergy has (drinker, beer, bar) triples such that the bar serves the beer, the drinker frequents the bar and likes the beer.

20

Example: The View

21

Pick one copy of each attribute

```
CREATE VIEW Synergy AS
    SELECT Likes.drinker, Likes.beer, Sells.bar
    FROM Likes, Sells, Frequents
    WHERE Likes.drinker = Frequents.drinker
        AND Likes.beer = Sells.beer
        AND Sells.bar = Frequents.bar;
```

Natural join of Likes, Sells, and Frequents

21

Updates on Views

22

- Generally, it is impossible to modify a virtual view, because it doesn't exist.
- Can't we "translate" updates on views into "equivalent" updates on base tables?
  - Not always (in fact, not often)
  - Most systems prohibit most view updates
- We cannot insert into Synergy --- it is a virtual view.

22

Interpreting a View Insertion

23

- But we could try to translate a (drinker, beer, bar) triple into three insertions of projected pairs, one for each of Likes, Sells, and Frequents.

23

Insertion

24

```
INSERT INTO LIKES VALUES(n.drinker, n.beer);
INSERT INTO SELLS(bar, beer) VALUES(n.bar, n.beer);
INSERT INTO FREQUENTS VALUES(n.drinker, n.bar);
```

- Sells.price will have to be NULL.
- There isn't always a unique translation.

24

Materialized Views

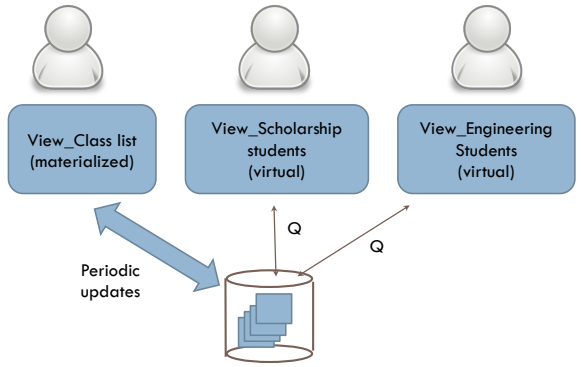
25

- *Materialized* = actually constructed and stored (keeping a temporary table)
- Concerns: maintaining correspondence between the base table and the view when the base table is updated
- Strategy: incremental update

25

Example

26



26

Example: Class Mailing List

27

- The class mailing list db3students is in effect a materialized view of the class enrollment
- Updated periodically
  - You can enroll and miss an email sent out after you enroll.
- Insertion into materialized view normally followed by insertion into base table

27

Materialized View Updates

28

- Update on a single view without aggregate operations: update may map to an update on the underlying base table (most SQL implementations)
- Views involving joins: an update *may map to an* update on the underlying base relations not always possible

28

Example: A Data Warehouse

29

- Wal-Mart stores every sale at every store in a database.
- Overnight, the sales for the day are used to update a *data warehouse* = materialized views of the sales.
- The warehouse is used by analysts to predict trends and move goods to where they are selling best.

29

Materialized Views

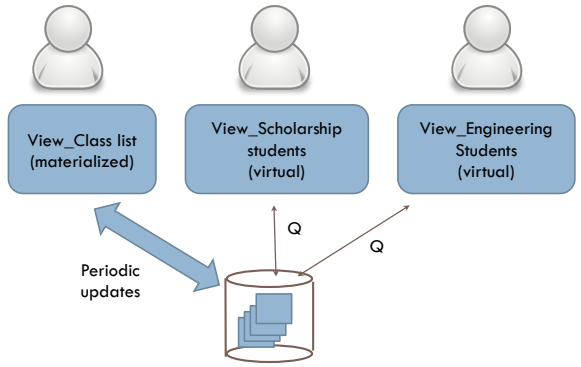
1

- *Materialized* = actually constructed and stored (keeping a temporary table)
- Concerns: maintaining correspondence between the base table and the view when the base table is updated
- Strategy: incremental update

1

Example

2



2

Example: Class Mailing List

3

- The class mailing list db3students is in effect a materialized view of the class enrollment
- Updated periodically
  - You can enroll and miss an email sent out after you enroll.
- Insertion into materialized view normally followed by insertion into base table

3

Materialized View Updates

4

- Update on a single view without aggregate operations: update may map to an update on the underlying base table (most SQL implementations)
- Views involving joins: an update *may map to an* update on the underlying base relations not always possible

4

Example: A Data Warehouse

5

- Wal-Mart stores every sale at every store in a database.
- Overnight, the sales for the day are used to update a *data warehouse* = materialized views of the sales.
- The warehouse is used by analysts to predict trends and move goods to where they are selling best.

5

INDEXES

6

Example

7

- Find the price of beers manufactured by Pete's and sold by Joe.

```
SELECT price
FROM Beers, Sells
WHERE manf = 'Pete''s' AND bar = 'Joe' AND
Sells.beer = Beers.name
```

7

An Index

8

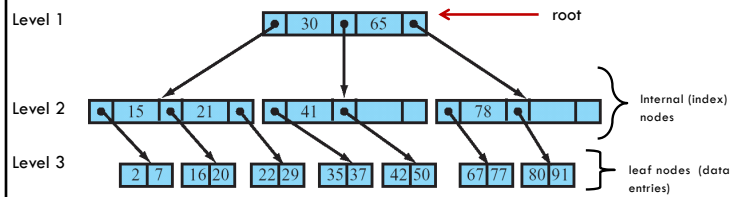
- A data structure used to speed access to tuples of a relation, based on values of one or more attributes ("search key" fields)
- Organizes records via trees or hashing
- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: use BeerInd (on manf) and SellInd (on bar, beer) to find the prices of beers manufactured by Pete's and sold by Joe.

8

B+ Tree Index

9

- The B+ tree structure is the most common index type in databases
- Index files can be quite large, often stored on disk, partially loaded into memory as needed
- Each node is at least 50% full

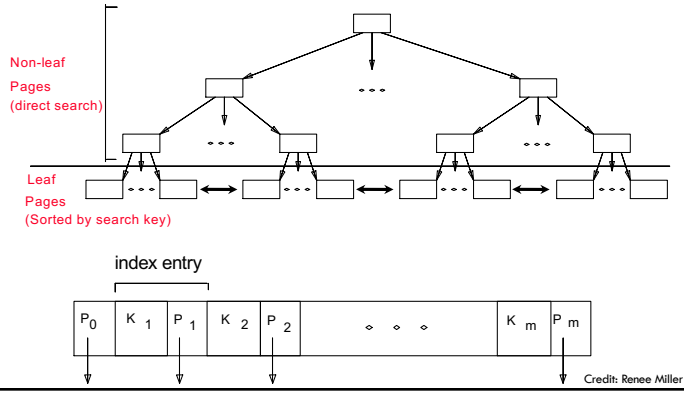


9

B+ Tree Index

10

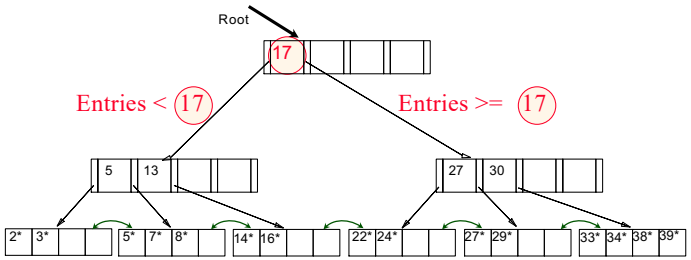
Supports equality and range-searches efficiently



10

Example

11

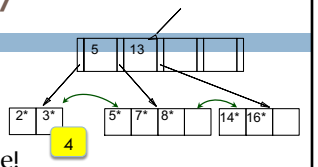


- Find 28*? 29*? All > 15* and < 30*
- Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

11

Inserting a Data Entry

12



- Find correct leaf L.
- Put data entry onto L.
  - If L has enough space, done!
  - Else, must ***split*** L (into L and a new node L2)
    - Redistribute entries evenly, copy up middle key.
    - Insert index entry pointing to L2 into parent of L.
- This can happen recursively
  - To split index node, redistribute entries evenly, but push up middle key.
- Splits "grow" tree; root split increases height.

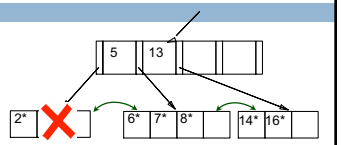
Insert data value 4

12

Deleting a Data Entry

13

Delete value 3



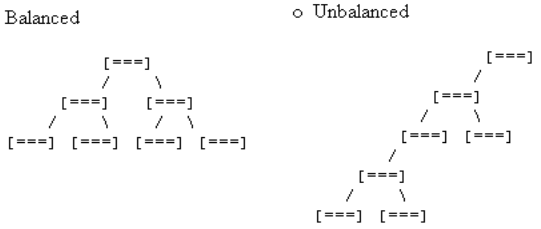
- Start at root, find leaf L where entry belongs.
- Remove the entry.
  - If L is at least half-full, done!
  - If not,
    - Try to re-distribute, borrowing from sibling (adjacent node with same parent as L).
    - If re-distribution fails, merge L and sibling.
- If merge occurred, must delete entry (pointing to L or sibling) from parent of L.
- Merge could propagate to root, decreasing height.

13

Balanced vs. Unbalanced Trees

14

- In a balanced tree, every path from the root to a leaf node is the same length.



Credit: S. Lee

14

Hash Based Indexes

15

- Good for equality selections.
- Index is a collection of buckets
  - Bucket = primary page plus zero or more overflow pages.
  - Buckets contain data entries.
- Hashing function h: h(r) = bucket in which (data entry for) record r belongs. h looks at the search key fields of r.
  - No need for "index entries" in this scheme.

15

Index Classification

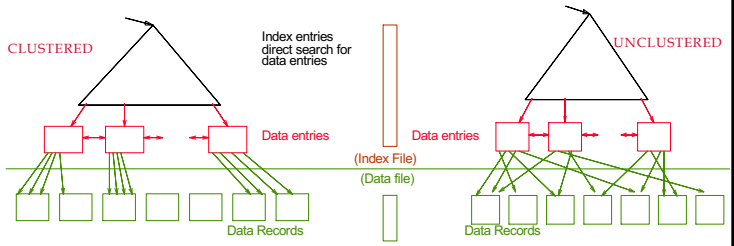
16

- Primary vs. secondary: If search key contains primary key, then called primary index.
  - Unique index: Search key contains a candidate key.
- Clustered vs. unclustered: If order of index data entries is the same as order of data records, then called clustered index.
  - A table can have at most one clustered index – why?

16

Clustered vs. Unclustered Index

17



17

Declaring Indexes

18

- No standard!
- Typical syntax:

```
CREATE INDEX BeerInd ON Beers(manf);
CREATE INDEX SellInd ON Sells(bar,
  beer);
```

18

Using Indexes

19

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: use BeerInd and SellInd to find the prices of beers manufactured by Pete's and sold by Joe.

19

Using Indexes --- (2)

20

```
SELECT price
FROM Beers, Sells
WHERE manf = 'Pete''s' AND
   Beers.name = Sells.beer AND
   bar = 'Joe''s Bar';
```

1. Use BeerInd to get all the beers made by Pete's.
2. Then use SellInd to get prices of those beers, with bar = 'Joe"s Bar'

20

Understanding the Workload

21

- For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
- For each update in the workload:
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

21

Choice of Indexes

22

- What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- For each index, what kind of an index should it be?
  - Clustered? Hash/tree?

22

Choice of Indexes (cont'd)

23

- One approach: Consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index.
  - Implies an understanding of how a DBMS evaluates queries and creates query evaluation plans.
- Before creating an index, must also consider the impact on updates in the workload!
  - Trade-off: Indexes can make queries go faster, updates slower. Require disk space, too.

23

Guidelines

24

- Attributes in WHERE clause are candidates for index keys.
  - Exact match condition suggests hash index.
  - Range query suggests tree index.
- Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
- Try to choose indexes that benefit as many queries as possible.
  - Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

24

Examples

25

- B+ tree index on E.age can be used to get qualifying tuples.

```
SELECT  E.dno
FROM  Emp E
WHERE  E.age>40
```

- Equality queries and duplicates:
  - Indexing on E.hobby

```
SELECT  E.dno
FROM  Emp E
WHERE  E.hobby='Stamps'
```

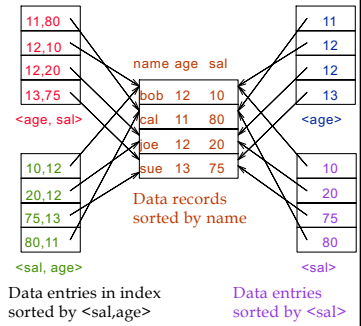
25

Composite Search Keys

26

- Composite Search Keys: Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. E.g. wrt <sal,age> index:
    - age=20 and sal =75
  - Range query:
    - age=20 and sal > 10
- Data entries in index sorted by search key to support range queries.

Examples of composite key indexes



26

Midterm – Thurs. Oct. 21st

2

- 50 minutes, 1:30-2:20pm
- Online
- Topics covered…

2

Relational Model

3

- Logical model, physical model
- Constraints, keys (superkey, PK, FK)
- Referential integrity, ways of enforcement

3

E-R Model

4

- Read and interpret an ER diagram
- How to translate English requirements to an ER diagram
- Different types of relationships
- Weak entities
- ISA hierarchies
- Constraints

4

SQL

5

- DDL, DML
- Relational predicates, clauses, operators, joins, aggregation, grouping, etc.
- Keys: PKs, FKs, referential integrity (ways of enforcement)
- Bag semantics vs. set semantics
- Given a schema:
  - Evaluate the results (output) of an SQL query
  - Translate English statement to an SQL query

5

Views

6

- View definition
- Distinction between virtual vs. materialized views
- Insertions and updates on views

6

Question 1

7

For each of the following statements, indicate whether they are true or false:

a) In SQL, there can be multiple primary key declarations in one create table statement. **False**

b) A relation R(A, B, C) may have at most three (minimal) keys (not superkeys). **True**

c) Let R be a bag over the attributes A, B . If A is a key for R, then R is necessarily a set. **True**

7

Question 1 (cont'd)

8

d) In SQL, there can be multiple unique (key) declarations in one create table statement. **True**

e) The value of any arithmetic operation involving a null value (e.g., '5-Null') is null. **True**

f) In SQL, DDL stands for Data Definition Language and DML stands for Data Management Language. **False**

g) A weak entity set has one or more many-many relationships to other (supporting) entity sets. **False**

h) An update to a virtual view must eventually be synchronized to its base tables. **False**

8

Question 2: Create an ER diagram modeling the same information. If the ER diagram cannot capture all dependencies, explain.

9

```
create table Books (ISBN   char(10) primary key,
                    author char(30) foreign key references Authors,
                    title  char(50),
                    qty    int)

create table Authors (name char(30) primary key,
                      institution char(30))

create table Borrowers (cardno  int primary key,
                        name  char(30))

create table Loans (cardno int foreign key references Borrowers,
                    isbn   char(10) foreign key references Books,
                    due    date,
   primary key (cardno,isbn,due))
```

Entity Set Books with attributes ISBN (Key), title, qty. No author attribute!
Entity set Authors with attributes name (key), institution.
Entity set Borrowers with attributes cardno(key), name.
Binary Relationship set Wrote between Books and Authors that is many:one (solid arrow on Author side).
Binary Relationship Loans between Books, Borrowers with attribute due date.

9

Question 3

- Product(maker, model, price)
- PC(model, speed)
- Printer(model, type)

10

- model is the primary key for all relations.
- The only possible values of type are "laser" and "ink-jet".
- Every PC model and every printer model is a Product model (that is, every PC or printer must be referenced in the relation Product).
- The price of a product should not be more than 10% higher than the average price of all products.

```
create table Product (
model integer not null primary key,
maker char(20),
price integer (check price =< (select avg(price)*1.10 from Product))
)

create table PC (
model integer not null primary key,
speed char(20),
model foreign key references Product
)

create table Printer (
model integer not null primary key,
type char(20),
check (type in \{'laser', 'ink-jet'\}),
model foreign key references Product
)
```

10

Question 3(b)

- Product(maker, model, price)
- PC(model, speed)
- Printer(model, type)

11

Write in SQL: Find makers from whom a combination (PC and printer) can be bought for less than $2000 .

```
SELECT distinct p.maker
FROM Product p
WHERE EXISTS (
   SELECT *
   FROM PC pc, Printer t, Product p1, Product p2
   WHERE p1.model = pc.model and p2.model = t.model and
         p1.price + p2.price < 2000 and p1.maker = p.maker and
         p2.maker = p.maker )
```

11

Question 3c

2

- Product(maker, model, price)
- PC(model, speed)
- Printer(model, type)

Write in SQL: For each maker, find the minimum and maximum price of a (PC, ink-jet printer) combination.

```
SELECT p1.maker, min(p1.price+p2.price), max(p1.price+p2.price)
FROM  Product p1, Product p2, PC pc, Printer t
WHERE  t.type = 'ink-jet' and p1.model = pc.model and p2.model =
          t.model and p1.maker=p2.maker
GROUP BY  p1.maker
```

2

Question 4

3

Given the instance of two relations:

R:

A	B
1	2
3	4
1	3

S:

B	C
1	3
2	4

a) What is the result of the following query:

```
SELECT DISTINCT R.A
FROM R
WHERE R.A NOT IN (SELECT DISTINCT S.B AS A
                  FROM S
                  WHERE S.B = S.C)
```

A
1
3

3

Question 4b

4

R:

A	B
1	2
3	4
1	3

S:

B	C
1	3
2	4

```
SELECT R.A, S.C, avg(R.B) as av
FROM R, S
WHERE R.B < 4
GROUP BY R.A, S.C
HAVING max(R.B) >= 2
```

A	R.B	S.B	C
1	2	1	3
1	2	2	4
3	4	1	3
3	4	2	4
1	3	1	3
1	3	2	4

A	R.B	S.B	C
1	2	1	3
1	3	1	3
1	2	2	4
1	3	2	4

A	C	av
1	3	2.5
1	4	2.5

4

Question 5

5

Consider the following CREATE TABLE definition:

```
CREATE TABLE Midterm
  (A INT NOT NULL,
   B INT NOT NULL,
   C INT NOT NULL,
   PRIMARY KEY (A),
   FOREIGN KEY (B) REFERENCES Midterm(A) ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (C) REFERENCES Midterm(A) ON DELETE CASCADE ON UPDATE RESTRICT)
```

Consider the following instance table Midterm:

A	B	C
4	3	3
3	4	3

a) What is the result of the following statement:

```
UPDATE Midterm
SET B = B+1
WHERE B in (SELECT A FROM Midterm)
```

Answer: Error, the FK constraint is violated

5

6

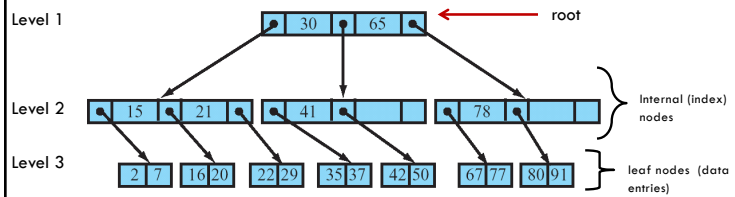
End

6

7

B+ Tree Index

- The B+ tree structure is the most common index type in databases
- Index files can be quite large, often stored on disk, partially loaded into memory as needed
- Each node is at least 50% full

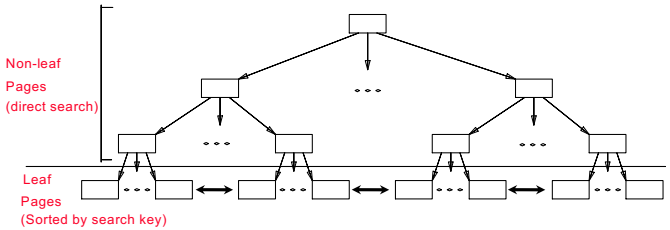


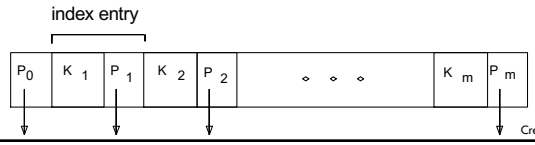
7

8

B+ Tree Index

Supports equality and range-searches efficiently

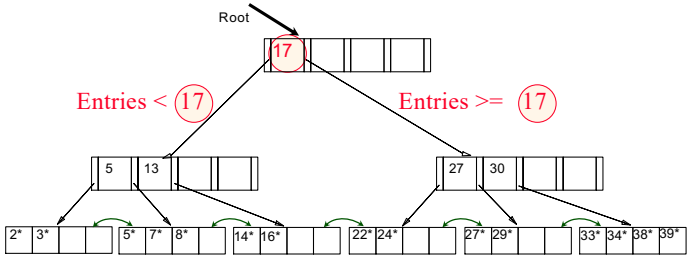




8

9

Example



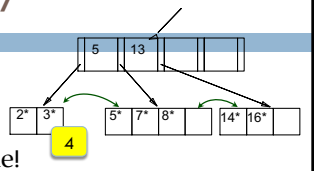
- Find 28*? 29*? All > 15* and < 30*
- Insert/delete: Find data entry in leaf, then change it. Need to adjust parent sometimes.
  - And change sometimes bubbles up the tree

9

Inserting a Data Entry

10

- Find correct leaf L.
- Put data entry onto L.
  - If L has enough space, done!
  - Else, must ***split*** L (into L and a new node L2)
    - Redistribute entries evenly, copy up middle key.
    - Insert index entry pointing to L2 into parent of L.
- This can happen recursively
  - To split index node, redistribute entries evenly, but push up middle key.
- Splits "grow" tree; root split increases height.



Insert data value 4

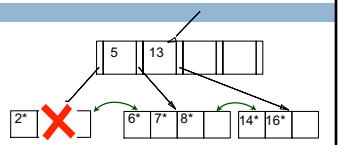
10

Deleting a Data Entry

Delete value 3

11

- Start at root, find leaf L where entry belongs.
- Remove the entry.
  - If L is at least half-full, done!
  - If not,
    - Try to re-distribute, borrowing from sibling (adjacent node with same parent as L).
    - If re-distribution fails, merge L and sibling.
- If merge occurred, must delete entry (pointing to L or sibling) from parent of L.
- Merge could propagate to root, decreasing height.

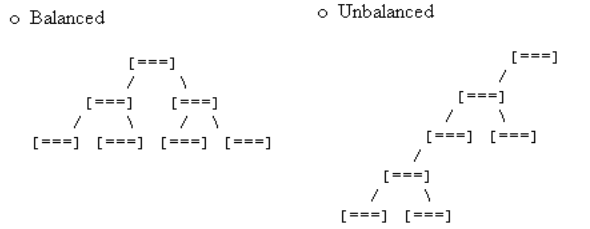


11

Balanced vs. Unbalanced Trees

12

- In a balanced tree, every path from the root to a leaf node is the same length.



Credit: S. Lee

12

Hash Based Indexes

13

- Good for equality selections.
- Index is a collection of buckets
  - Bucket = primary page plus zero or more overflow pages.
  - Buckets contain data entries.
- Hashing function h: h(r) = bucket in which (data entry for) record r belongs. h looks at the search key fields of r.
  - No need for "index entries" in this scheme.

13

Index Classification

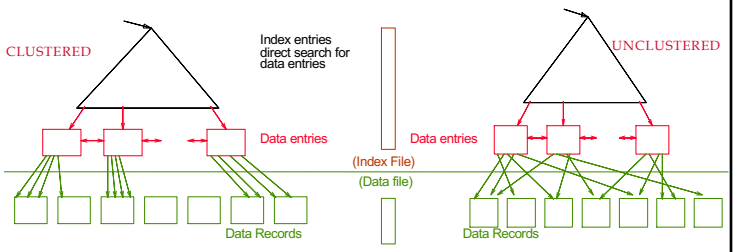
12

- Primary vs. secondary: If search key contains primary key, then called primary index.
  - Unique index: Search key contains a candidate key.
- Clustered vs. unclustered: If order of index data entries is the same as order of data records, then called clustered index.
  - A table can have at most one clustered index – why?

12

Clustered vs. Unclustered Index

13



13

Declaring Indexes

14

- No standard!
- Typical syntax:

```
CREATE INDEX BeerInd ON Beers(manf);
CREATE INDEX SellInd ON Sells(bar,
  beer);
```

14

Using Indexes

15

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: use BeerInd and SellInd to find the prices of beers manufactured by Pete's and sold by Joe.

15

Using Indexes --- (2)

16

```
SELECT price
FROM Beers, Sells
WHERE manf = 'Pete''s' AND
    Beers.name = Sells.beer AND
    bar = 'Joe''s Bar';
```

1. Use BeerInd to get all the beers made by Pete's.
2. Then use SellInd to get prices of those beers, with bar = 'Joe''s Bar'

16

Understanding the Workload

17

- For each query in the workload:
  - Which relations does it access?
  - Which attributes are retrieved?
  - Which attributes are involved in selection/join conditions? How selective are these conditions likely to be?
- For each update in the workload:
  - The type of update (INSERT/DELETE/UPDATE), and the attributes that are affected.

17

Choice of Indexes

18

- What indexes should we create?
  - Which relations should have indexes? What field(s) should be the search key? Should we build several indexes?
- For each index, what kind of an index should it be?
  - Clustered? Hash/tree?

18

Choice of Indexes (cont'd)

19

- One approach: Consider the most important queries in turn. Consider the best plan using the current indexes, and see if a better plan is possible with an additional index.
  - Implies an understanding of how a DBMS evaluates queries and creates query evaluation plans.
- Before creating an index, must also consider the impact on updates in the workload!
  - Trade-off: Indexes can make queries go faster, updates slower. Require disk space, too.

19

Guidelines

20

- Attributes in WHERE clause are candidates for index keys.
  - Exact match condition suggests hash index.
  - Range query suggests tree index.
- Multi-attribute search keys should be considered when a WHERE clause contains several conditions.
- Try to choose indexes that benefit as many queries as possible.
  - Since only one index can be clustered per relation, choose it based on important queries that would benefit the most from clustering.

20

Examples

21

- B+ tree index on E.age can be used to get qualifying tuples.

```
SELECT  E.dno
FROM  Emp E
WHERE  E.age>40
```

- Equality queries and duplicates:
  - Indexing on E.hobby

```
SELECT  E.dno
FROM  Emp E
WHERE  E.hobby='Stamps'
```

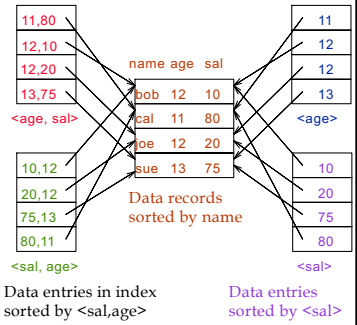
21

Composite Search Keys

22

- Composite Search Keys: Search on a combination of fields.
  - Equality query: Every field value is equal to a constant value. E.g. wrt <sal,age> index:
    - age=20 and sal =75
  - Range query:
    - age=20 and sal > 10
- Data entries in index sorted by search key to support range queries.

Examples of composite key indexes



22

Database Tuning

23

- A major problem in making a database run fast is deciding which indexes to create.
- Pro: An index speeds up queries that can use it.
- Con: An index slows down all modifications on its relation because the index must be modified too.

23

Example: Tuning

24

- Suppose the only things we did with our beers database was:
  1. Insert new beers into a relation (10%).
  2. Find the price of a given beer at a given bar (90%).
- Then SellInd on Sells(bar, beer) would be helpful, but BeerInd on Beers(manf) would be harmful.

24

Tuning Advisors

25

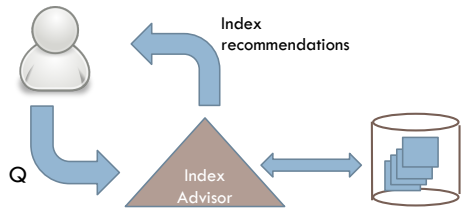
- A major research area
  - Because hand tuning is so hard.
- An advisor gets a *query load*, e.g.:
  1. Choose random queries from the history of queries run on the database, or
  2. Designer provides a sample workload.

25

Tuning Advisors --- (2)

26

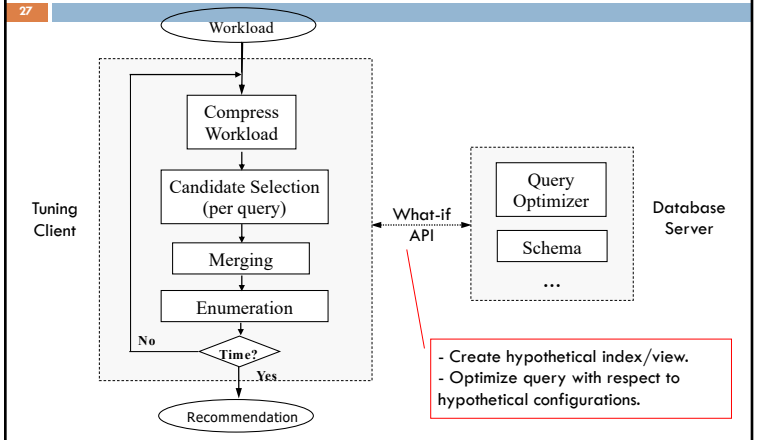
- The advisor generates candidate indexes and evaluates each on the workload.
  - Measure the improvement/degradation in the average running time of the queries.



26

Example: Database Tuning Architecture

27



27

Summary

28

- If selection queries are frequent, sorting the file or building an index is important.
  - Hash-based indexes only good for equality search.
  - Tree-based indexes best for range search; also good for equality search.

28

Summary (cont'd)

29

- Can have several indexes on a given file of data records, each with a different search key.
- Understanding the nature of the workload for the application
  - What are the important queries and updates? What attributes/relations are involved?
- Indexes are chosen to speed up important queries (and perhaps some updates!).
  - Consider index maintenance overhead on updates to key fields.
  - Choose indexes that can help many queries, if possible.

29

Relational Query Languages

2

- *Query languages*: Allow manipulation and retrieval of data from a database.
- Relational model supports simple, powerful QLs:
  - Formal foundation based on logic.
  - Allows for optimization.
- Query Languages != programming languages!
  - QLs not intended to be used for complex calculations.
  - QLs support easy, efficient access to large data sets.

Credit: R. Ramakrishnan, J. Gehrke

2

DBMS Architecture

3

How does a SQL engine work ?

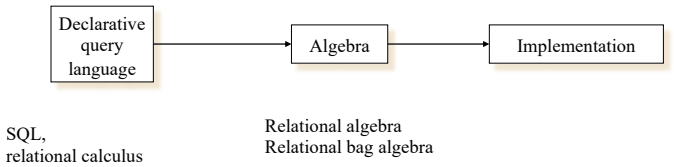
- SQL query → relational algebra plan
- Relational algebra plan → Optimized plan
- Execute each operator of the plan

3

Relational Algebra

4

- Formalism for creating new relations from existing ones
- Its place in the big picture:



4

Formal Relational Query Languages

5

Two mathematical Query Languages form the basis for SQL, and for implementation:

- *Relational Algebra*: More operational, very useful for representing execution plans.
- *Relational Calculus*: Lets users describe what they want, rather than how to compute it. (Non-operational, *declarative*.)

5

What is an "Algebra"

6

- Mathematical system consisting of:
  - *Operands* --- variables or values from which new values can be constructed.
  - *Operators* --- symbols denoting procedures that construct new values from given values.

Credit: Renee J. Miller

6

What is Relational Algebra?

7

- An algebra whose operands are relations or variables that represent relations.
- Operators are designed to do the most common things that we need to do with relations in a database.
  - The result is an algebra that can be used as a *query language* for relations.

7

Core Relational Algebra

8

- Union, intersection, and difference.
  - Usual operations, but *both operands must have the same relation schema*.
- Selection: picking certain rows.
- Projection: picking certain columns.
- Products and joins: compositions of relations.
- Renaming of relations and attributes.

Since each operation returns a relation, operations can be *composed*

8

Selection

9

- R1 := σ_C(R2)
  - C is a condition (as in "if" statements) that refers to attributes of R2.
  - R1 is all those tuples of R2 that satisfy C.

9

Example: Selection

10

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoeMenu := $\sigma_{bar="Joe's"}(Sells)$:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75

- Selects rows that satisfy *selection condition*.
- *Schema* of result identical to schema of (only) input relation.
- *Result* relation can be the *input* for another relational algebra operation. (*Operator composition*.)

10

Projection

11

- R1 := π_L (R2)
  - *L* is a list of attributes from the schema of R2.
  - R1 is constructed by looking at each tuple of R2, extracting the attributes on list *L*, in the order specified, and creating from those components a tuple for R1.

11

Example: Projection

12

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

Prices := $\pi_{beer,price}(Sells)$:

beer	price
Bud	2.50
Bud	2.50
Miller	2.75
Miller	3.00

Schema of result contains exactly the fields in the projection list, with the same names that they had in the (only) input relation.

12

Example

13

Relation Sells:

bar	beer	price
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Miller	3.00

JoePrices := $\pi_{beer,price}(\sigma_{bar="Joe's"}(Sells))$

Beer	Price
Bud	2.50
Miller	2.75

13

Extended Projection

14

- Using the same Π_L operator, we allow the list L to contain arbitrary expressions involving attributes:
  - Arithmetic on attributes, e.g., $A+B \rightarrow C$.

14

Example: Extended Projection

15

R =

A	B
1	2
3	4

$\Pi_{A+B \rightarrow C, A \rightarrow A1, A \rightarrow A2}$ (R) =

C	A1	A2
3	1	1
7	3	3

15

Product

16

- R3 := R1 X R2
  - Pair each tuple t1 of R1 with each tuple t2 of R2.
  - Concatenation t1t2 is a tuple of R3.
  - Schema of R3 is the attributes of R1 and then R2, in order.
  - But beware attribute *A* of the same name in R1 and R2:
    - In relational algebra use renaming to distinguish
    - in SQL use R1.*A* and R2.*A*.

16

Example: R3 := R1 X R2

17

R1(

A	B
1	2
3	4

)

R2(

B	C
5	6
7	8
9	10

)

R3(

A,	R1.B,	R2.B,	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

)

17

Theta-Join

18

- R3 := R1 $\bowtie_C$ R2
  - Take the product R1 X R2.
  - Then apply σ_C to the result.
- As for σ_C can be any boolean-valued condition.
  - A θ B, where θ is =, <, etc.; hence the name "theta-join."

18

Example: Theta Join

19

Sells(

bar,	beer,	price)
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

Bars(

name,	addr)
Joe's	Maple St.
Sue's	River Rd.

BarInfo := Sells $\bowtie_{Sells.bar = Bars.name}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr)
Joe's	Bud	2.50	Joe's	Maple St.
Joe's	Miller	2.75	Joe's	Maple St.
Sue's	Bud	2.50	Sue's	River Rd.
Sue's	Coors	3.00	Sue's	River Rd.

19

Natural Join

20

- A useful join variant (*natural* join) connects two relations by:
  - Equating attributes of the same name, and
  - Projecting out one copy of each pair of equated attributes.
- Denoted R3 := R1 $\bowtie$ R2.

20

Example: Natural Join

21

Sells(

bar,	beer,	price)
Joe's	Bud	2.50
Joe's	Miller	2.75
Sue's	Bud	2.50
Sue's	Coors	3.00

Bars(

bar,	addr)
Joe's	Maple St.
Sue's	River Rd.

BarInfo := Sells $\bowtie$ Bars

Note: Bars.name has become Bars.bar to make the natural join non-trivial

BarInfo(

bar,	beer,	price,	addr)
Joe's	Bud	2.50	Maple St.
Joe's	Milller	2.75	Maple St.
Sue's	Bud	2.50	River Rd.
Sue's	Coors	3.00	River Rd.

21

Renaming

22

- The ρ operator gives a new schema to a relation.
- R1 := $\rho_{R1(A1,\ldots,An)}$(R2) makes R1 be a relation with attributes A1,…,A*n* and the same tuples as R2.
- Simplified notation: R1(A1,…,A*n*) := R2.

22

Example: Renaming

23

Bars(

name,	addr
Joe's	Maple St.
Sue's	River Rd.

)

R(bar, addr) := Bars

R(

bar,	addr
Joe's	Maple St.
Sue's	River Rd.

)

23

Set Operators

24

- Union, Intersection and Difference are defined only for **union compatible** relations.
- Two relations are union compatible if they have the same set of attributes and the types (domains) of the attributes are the same.
- E.g., two relations that are not union compatible:
  - **Student (sNumber, sName)**
  - **Course (cNumber, cName)**

24

Relational Algebra on Bags

25

- A *bag* (or *multiset*) is like a set, but an element may appear more than once.
- Example: {1,2,1,3} is a bag.
- Example: {1,2,3} is also a bag that happens to be a set.

25

Union: ∪

26

- Consider two bags R_1 and R_2 that are union-compatible. Suppose a tuple t appears in R_1 m times, and in R_2 n times. Then in the union, t appears $m + n$ times.

R_1

A	B
1	2
3	4
1	2

R_2

A	B
1	2
3	4
5	6

$R_1 \cup R_2$

A	B
1	2
1	2
1	2
3	4
3	4
5	6

26

Intersection: ∩

27

- Consider two bags R_1 and R_2 that are union-compatible. Suppose a tuple t appears in R_1 m times, and in R_2 n times. Then in the intersection, t appears min (m, n) times.

R_1

A	B
1	2
3	4
1	2

R_2

A	B
1	2
3	4
5	6

$R_1 \cap R_2$

A	B
1	2
3	4

27

Difference: -

28

- Consider two bags R_1 and R_2 that are union-compatible. Suppose a tuple t appears in R_1 m times, and in R_2 n times. Then in $R_1 - R_2$, t appears max (0, m - n) times.

R_1

A	B
1	2
3	4
1	2

R_2

A	B
1	2
3	4
5	6

$R_1 - R_2$

A	B
1	2

28

Building Complex Expressions

29

- Combine operators with parentheses and precedence rules.
- Three notations, just as in arithmetic:
  - Sequences of assignment statements.
  - Expressions with several operators.
  - Expression trees.

Credit: Renee J. Miller

29

Sequences of Assignments

30

- Create temporary relation names.
- Renaming can be implied by giving relations a list of attributes.
  - $\pi_{A+B->C,A->A1,A->A2}$ (R)
- Example: R3 := R1 $\bowtie_C$ R2 can be written:

  R4 := R1 X R2

  R3 := σ_C (R4)

30

Expressions in a Single Assignment

31

Example: the theta-join R3 := R1 $\bowtie_C$ R2 can be written as

- R3 := σ_C (R1 X R2)

- Precedence of relational operators: (parentheses supercedes)
  - [σ, π, ρ] (highest).
  - [X, $\bowtie$].
  - $\cap$.
  - [$\cup$, —]

31

Expression Trees

32

- Leaves are operands --- either variables standing for relations or particular, constant relations.
- Interior nodes are operators, applied to their child or children.

32

Example: Tree for a Query

33

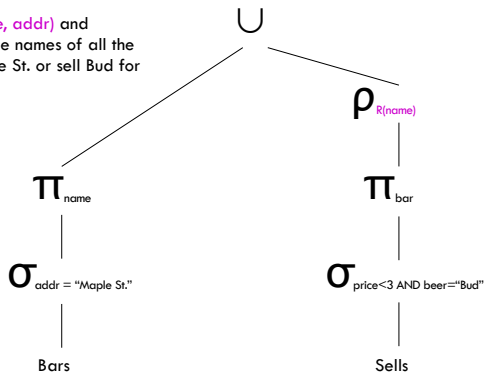
- Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.

33

As a Tree:

34

Using the relations Bars(name, addr) and Sells(bar, beer, price), find the names of all the bars that are either on Maple St. or sell Bud for less than $3.



34

Example: Self-Join

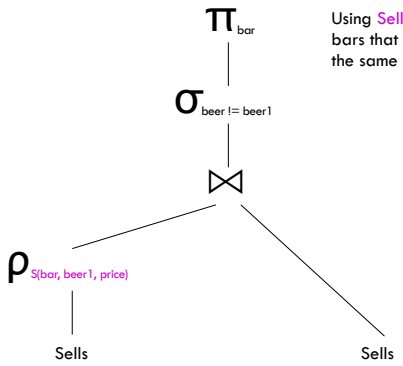
35

- Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.
- Strategy: by renaming, define a copy of Sells, called S(bar, beer1, price). The natural join of Sells and S consists of quadruples (bar, beer, beer1, price) such that the bar sells both beers at this price.

35

The Tree

36



Using Sells(bar, beer, price), find the bars that sell two different beers at the same price.

36

Schemas for Results

37

- Union, intersection, and difference: the schemas of the two operands must be the same, so use that schema for the result.
- Selection: schema of the result is the same as the schema of the operand.
- Projection: list of attributes tells us the schema.

37

Schemas for Results

38

- Product: schema is the attributes of both relations.
  - Distinguish two attributes with the same name.
- Theta-join: same as product.
- Natural join: union of the attributes of the two relations. Keep only one copy of the equated attributes.
- Renaming: the operator tells the schema.

38

Lecture Example

39

39

The Extended Algebra

40

δ = eliminate duplicates from bags.

τ = sort tuples.

γ = grouping and aggregation.

Outerjoin : avoids "dangling tuples" = tuples that do not join with anything.

Credit: Renee J. Miller

40

Duplicate Elimination

41

- R1 := δ(R2).
- R1 consists of one copy of each tuple that appears in R2 one or more times.

41

Example: Duplicate Elimination

42

R =

A	B
1	2
3	4
1	2

δ(R) =

A	B
1	2
3	4

42

Sorting

43

- R1 := τ_L (R2).
  - *L* is a list of some of the attributes of R2.
- R1 is the list of tuples of R2 sorted first on the value of the first attribute on *L*, then on the second attribute of *L*, and so on.
  - Break ties arbitrarily.

43

Example: Sorting

44

R =

A	B
1	2
3	4
5	2

T_B (R) =

A	B
5	2
1	2
3	4

44

Aggregation Operators

45

- Aggregation operators are not formally operators of relational algebra.
- Rather, they apply to entire columns of a table and produce a single result.
- The most important examples: SUM, AVG, COUNT, MIN, and MAX.

45

Example: Aggregation

46

R =

A	B
1	3
3	4
3	2

SUM(A) = 7
COUNT(A) = 3
MAX(B) = 4
AVG(B) = 3

46

Grouping Operator

47

- R1 := γ_L (R2). L is a list of elements that are either:
  1. Individual (*grouping*) attributes.
  2. AGG(A), where AGG is one of the aggregation operators and A is an attribute.
     - An arrow and a new attribute name renames the component.

47

Applying $\gamma_L(R)$

48

- Group *R* according to all the grouping attributes on list *L*.
  - That is: form one group for each distinct list of values for those attributes in *R*.
- Within each group, compute AGG(*A*) for each aggregation on list *L*.
- Result has one tuple for each group:
  1. The grouping attributes and
  2. The group's aggregations.

48

Example: Grouping/Aggregation

49

R =

A	B	C
1	2	3
4	5	6
1	2	5

$\gamma_{A,B,AVG(C)\rightarrow X}(R)$ = ??

First, group *R* by *A* and *B* :

A	B	C
1	2	3
1	2	5
4	5	6

Then, average C within groups:

A	B	X
1	2	4
4	5	6

49

Recall: Outerjoin

50

- Suppose we join $R \bowtie_C S$.
- A tuple of *R* that has no tuple of *S* with which it joins is said to be *dangling*.
  - Similarly for a tuple of *S*.
- Outerjoin preserves dangling tuples by padding them NULL.

50

Example: Outerjoin

51

R =

A	B
1	2
4	5

S =

B	C
2	3
6	7

(1,2) joins with (2,3), but the other two tuples are dangling.

R FULL OUTERJOIN S =

A	B	C
1	2	3
4	5	NULL
NULL	6	7

51

Outer Join – Example

52

- instructor

ID	name	dept_name
10101 12121 15151	Srinivasan Wu Mozart	Comp. Sci. Finance Music

teaches

ID	course_id
10101 12121 76766	CS-101 FIN-201 BIO-101

instructor ⋈ *teaches*

ID	name	dept_name	course_id
10101 12121	Srinivasan Wu	Comp. Sci. Finance	CS-101 FIN-201

- Left Outer Join

instructor ⟕ *teaches*

ID	name	dept_name	course_id
10101 12121 15151	Srinivasan Wu Mozart	Comp. Sci. Finance Music	CS-101 FIN-201 *NULL*

52

Outer Join – Example

53

- instructor

ID	name	dept_name
10101 12121 15151	Srinivasan Wu Mozart	Comp. Sci. Finance Music

teaches

ID	course_id
10101 12121 76766	CS-101 FIN-201 BIO-101

- *instructor* ⟖ *teaches*

ID	name	dept_name	course_id
10101 12121 76766	Srinivasan Wu null	Comp. Sci. Finance null	CS-101 FIN-201 BIO-101

- *instructor* ⟗ *teaches*

ID	name	dept_name	course_id
10101 12121 15151 76766	Srinivasan Wu Mozart null	Comp. Sci. Finance Music null	CS-101 FIN-201 *null* BIO-101

53

Operations on Bags

54

A **bag** = a set with repeated elements

All operations need to be defined carefully on bags

- {a,b,b,c}∪{a,b,b,b,e,f,f}={a,a,b,b,b,b,b,c,e,f,f}

- $\sigma_C(R)$: preserve the number of occurrences
- $\Pi_A(R)$: no duplicate elimination
- Cartesian product, join: no duplicate elimination

Important ! Relational Engines work on bags, not sets !

54

Why Bags?

55

- SQL, the most important query language for relational databases, is actually a bag language.
- Some operations, like projection, are more efficient on bags than sets.

55

Operations on Bags

56

- Selection applies to each tuple, so its effect on bags is like its effect on sets.
- Projection also applies to each tuple, we do not eliminate duplicates.
- Products and joins are done on each pair of tuples, so duplicates in bags have no effect on how we operate.

56

Example: Bag Selection

57

R(

A,	B
1	2
5	6
1	2

)

$\sigma_{A+B<5}(R)$ =

A	B
1	2
1	2

57

Example: Bag Projection

58

R(

A,	B
1	2
5	6
1	2

)

$\pi_A(R)$ =

A
1
5
1

58

Example: Bag Product

59

R(

A,	B
1	2
5	6
1	2

)

S(

B,	C
3	4
7	8

)

R X S =

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	3	4
5	6	7	8
1	2	3	4
1	2	7	8

59

Example: Bag Theta-Join

60

R(

A,	B)
1	2
5	6
1	2

S(

B,	C)
3	4
7	8

$R \bowtie_{R.B<S.B} S =$

A	R.B	S.B	C
1	2	3	4
1	2	7	8
5	6	7	8
1	2	3	4
1	2	7	8

60

Bag Union

61

- An element appears in the union of two bags the sum of the number of times it appears in each bag.
- Example: $\{1,2,1\} \cup \{1,1,2,3,1\} = \{1,1,1,1,1,2,2,3\}$

61

Bag Intersection

62

- An element appears in the intersection of two bags the minimum of the number of times it appears in either bag
- Example: $\{1,2,1,1\} \cap \{1,2,1,3\} = \{1,1,2\}$.

62

Bag Difference

63

- An element appears in the difference $A - B$ of bags as many times as it appears in A, minus the number of times it appears in B.
- Example: $\{1,2,1,1\} - \{1,2,3\} = \{1,1\}$.

63

Beware: Bag Laws != Set Laws

64

- Some, but *not all* algebraic laws that hold for sets also hold for bags.
- Example: the commutative law for union ($R \cup S = S \cup R$) *does* hold for bags.
  - Since addition is commutative, adding the number of times x appears in R and S doesn't depend on the order of R and S.

64

Example: A Law That Fails

65

- Set union is *idempotent*, meaning that $S \cup S = S$.
- However, for bags, if x appears n times in S, then it appears $2n$ times in $S \cup S$.
- Thus $S \cup S$!= S in general.
  - e.g., {1} ∪ {1} = {1,1} != {1}.

What about Intersection?

65

Lecture Example

66

66

DESIGN THEORY FOR RELATIONAL DATABASES

1

Introduction

2

- There are always many different schemas for a given set of data.
- E.g., you could combine or divide tables.
- How do you pick a schema? Which is better? What does "better" mean?
- Fortunately, there are some principles to guide us.

2

Schemas and Constraints

3

- Consider the following sets of schemas:

  Students(macid, name, email)

  vs.

  Students(macid, name)
  Emails(macid, address)

- Consider also:

  House(street, city, value, owner, propertyTax)

  vs.

  House(street, city, value, owner)
  TaxRates(city, value, propertyTax)

Constraints are domain-dependent

Acknowledgements: R. J. Miller, M. Papagelis

3

Avoid redundancy

4

This table has redundant data, and that can lead to anomalies.

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

- Update anomaly: if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- Deletion anomaly: If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

4

Database Design Theory

5

- It allows us to improve a schema systematically.
- General idea:
  - Express constraints on the data
  - Use these to decompose the relations
- Ultimately, get a schema that is in a "normal form" that guarantees good properties, such as no anomalies.
- "Normal" in the sense of conforming to a standard.
- The process of converting a schema to a normal form is called normalization.

5

Part I: Functional Dependency Theory

6

6

Keys

7

- *K* is a *key* for *R* if *K* uniquely determines all of *R*, and no proper subset of *K* does.
- *K* is a *superkey* for relation *R* if *K* contains a key for *R*.
  ("superkey" is short for "superset of key".)

7

Example

8

RegNum	Surname	FirstName	BirthDate	DegreeProg
284328	Smith	Luigi	29/04/59	Computing
296328	Smith	John	29/04/59	Computing
587614	Smith	Lucy	01/05/61	Engineering
934856	Black	Lucy	01/05/61	Fine Art
965536	Black	Lucy	05/03/58	Fine Art

- **RegNum** is a key: i.e., **RegNum** is a superkey and it contains a sole attribute, so it is minimal.
- **{Surname, Firstname, BirthDate}** is another key

8

Functional Dependencies

9

- Need a special type of constraint to help us with normalization
- X →Y is an assertion about a relation *R* that whenever two tuples of *R* agree on all the attributes in set *X*, they must also agree on all attributes in set *Y*.
- E.g., suppose X = {AB}, Y = {C}

R

A	B	C
x1	y1	c2
x1	y1	c2
x2	y2	c3
x2	y2	c3

9

Functional Dependencies

10

- Say "X →Y holds in *R*."
  "X functionally determines Y."
- Convention: …, *X*, *Y*, *Z* represent sets of attributes; *A*, *B*, C,… represent single attributes.
- Convention: no braces used for sets of attributes, just *ABC*, rather than {*A*,*B*,C }.

10

Why "functional dependency"?

11

- "dependency" because the value of Y depends on the value of X.
- "functional" because there is a mathematical function that takes a value for X and gives a *unique* value for Y.

11

Properties about FDs

12

- Rules
  - Splitting/combining
  - Trivial FDs
  - Armstrong's Axioms
- Algorithms related to FDs
  - the closure of a set of attributes of a relation
  - a minimal basis of a relation

12

Splitting Right Sides of FDs

13

- $X \rightarrow A_1A_2\ldots A_n$ holds for R exactly when each of $X \rightarrow A_1$, $X \rightarrow A_2$,…, $X \rightarrow A_n$ hold for R.
- Example: $A \rightarrow BC$ is equivalent to $A \rightarrow B$ and $A \rightarrow C$.
- Combining: if $A \rightarrow F$ and $A \rightarrow G$, then $A \rightarrow FG$

- There is no splitting rule for the left side
  - $ABC \rightarrow DEF$ is NOT the same as $AB \rightarrow DEF$ and $C \rightarrow DEF$!
- We'll generally express FDs with singleton right sides.

13

Example: FDs

14

Drinkers(name, addr, beersLiked, manf, favBeer)

Reasonable FDs to assert:

- name → addr, favBeer.
  - Note this FD is the same as: name → addr and name → favBeer.
- beersLiked → manf

14

Example: Possible Data

15

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

Because name → addr

Because name → favBeer

Because beersLiked → manf

15

Trivial FDs

16

- Not all functional dependencies are useful
  - A → A always holds
  - ABC → A also always holds (right side is subset of left side)
- FD with an attribute on both sides
  - ABC → AD becomes ABC → D
  - Or, in singleton form, delete trivial FDs
    ABC → A and ABC → D becomes just ABC → D

16

Superkey

17

Drinkers(name, addr, beersLiked, manf, favBeer)

- {name, beersLiked} is a superkey because together these attributes determine all the other attributes.
  - name → addr, favBeer
  - beersLiked → manf

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

17

Example: Key

18

- {name, beersLiked} is a key because neither {name} nor {beersLiked} is a key on its own.
  - name doesn't → manf; beersLiked doesn't → addr.
- There are no other keys, but lots of superkeys.
  - Any superset of {name, beersLiked}.

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

18

FDs are a generalization of keys

19

- Functional dependency: X → Y
- Superkey: X → R
- A superkey must include all the attributes of the relation on the RHS.
- An FD can involve just a subset of them
  - Example:

Houses (street, city, value, owner, tax)

- street,city → value, owner, tax *(both FD and key)*
- city,value → tax *(FD only)*

19

Identifying functional dependencies

20

- FDs are domain knowledge
  - Intrinsic features of the data you're dealing with
  - Something you know (or assume) about the data
- Database engine cannot identify FDs for you
  - Designer must specify them as part of schema
  - DBMS can only enforce FDs when told to
- DBMS cannot "optimize" FDs either
  - It has only a finite sample of the data
  - An FD constrains the entire domain

20

Coincidence or FD?

21

ID	Email	City	Country	Surname
1983	tom@gmail.com	Bern	Switzerland	Mendes
8624	jones@bell.com	London	Canada	Jones
9141	scotty@gmail.com	Winnipeg	Canada	Jones
1204	birds@gmail.com	Aachen	Germany	Lakemeyer

- In this instance:
  - Surname → Country
  - City → Country
- Are these FDs?

21

Coincidence or FD

22

- We have an FD only if it holds for *every* instance of the relation.
- You can't know this just by looking at one instance.
- You can only determine this based on knowledge of the domain.

22

Armstrong's Axioms

23

X, Y, Z are sets of attributes

1. **Reflexivity:** If $Y \subseteq X$, then $X \rightarrow Y$
2. **Augmentation:** If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
3. **Transitivity:** If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$
4. **Union:** If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
5. **Decomposition:** If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$

23

Inferring FDs

24

- Given a set of FDs, we can often infer further FDs.
- This will come in handy when we apply FDs to the problem of database design.

24

Dependency Inference

25

- Suppose we are given FDs
  $X_1 \rightarrow A_1$,
  $X_2 \rightarrow A_2$,
  …,
  $X_n \rightarrow A_n$.
- Does the FD $Y \rightarrow B$ also hold in any relation that satisfies the given FDs?
- Example: If $A \rightarrow B$ and $B \rightarrow C$ hold, surely $A \rightarrow C$ holds, even if we don't say so.
  $A \rightarrow C$ is ***entailed (implied)*** by $\{A \rightarrow B, B \rightarrow C\}$

25

Transitive Property

26

The transitive property holds for FDs

- Consider the FDs: $A \rightarrow B$ and $B \rightarrow C$; then $A \rightarrow C$ holds
- Consider the FDs: $AD \rightarrow B$ and $B \rightarrow CD$; then $AD \rightarrow CD$ holds or just $AD \rightarrow C$ (because of trivial FDs)

26

Method 1: Prove it from first principles

27

- To test if $Y \rightarrow B$, start by assuming two tuples agree on all attributes of Y.

←Y →

t1: aaaaa bb. . . b

t2: aaaaa ?? . . . ?

27

Example

28

ClientID	Income	OtherProd	Rate	Country	City	State
225	High	A	2.1%	USA	San Francisco	MD
420	High	A	2.1%	USA	San Francisco	CA
333	High	B	3.0%	USA	San Francisco	CA
576	High	B	3.0%	USA	San Francisco	CA
128	Low	C	4.5%	UK	Reading	Berkshire
193	Low	C	4.5%	UK	London	London
550	Low	B	3.5%	UK	London	London

F1: [Income, OtherProd] → [Rate] F2: [Country, City] → [State]

How to prove it in the general case?

28

Closure Test for FDs

29

- Given attribute set Y and FD set F
  - Denote Y_F^+ or Y^+ the closure of Y relative to F

    Y_F^+ = set of all FDs given or implied by Y
- Computing the closure of Y
  - Start: $Y_F^+ = Y$, $F' = F$
  - While there exists an $f \in F'$ s.t. LHS(f) $\subseteq Y_F^+$:

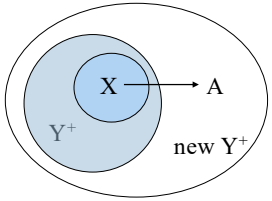
    $Y_F^+ = Y_F^+ \cup RHS(f)$

    $F' = F' - f$
  - At end: $Y \rightarrow B$ for all $B \in Y_F^+$

Acknowledgements: M. Papagelis

29

Computing the closure Y^+ of a set of attributes Y
Given FDs F:



30

Inferring FDs

25

- Given a set of FDs, we can often infer further FDs.
- This will come in handy when we apply FDs to the problem of database design.

25

Dependency Inference

26

- Suppose we are given FDs
  $X_1 \rightarrow A_1$,
  $X_2 \rightarrow A_2$,
  …,
  $X_n \rightarrow A_n$.
- Does the FD $Y \rightarrow B$ also hold in any relation that satisfies the given FDs?
- Example: If $A \rightarrow B$ and $B \rightarrow C$ hold, surely $A \rightarrow C$ holds, even if we don't say so.
  $A \rightarrow C$ is ***entailed (implied)*** by $\{A \rightarrow B, B \rightarrow C\}$

26

Transitive Property

27

The transitive property holds for FDs

- Consider the FDs: $A \rightarrow B$ and $B \rightarrow C$; then $A \rightarrow C$ holds
- Consider the FDs: $AD \rightarrow B$ and $B \rightarrow CD$; then $AD \rightarrow CD$ holds or just $AD \rightarrow C$ (because of trivial FDs)

27

Method 1: Prove it from first principles

28

- To test if $Y \rightarrow B$, start by assuming two tuples agree on all attributes of Y.

←Y →

t1: aaaaa bb. . . b

t2: aaaaa ?? . . . ?

28

Example

29

ClientID	Income	OtherProd	Rate	Country	City	State
225	High	A	2.1%	USA	San Francisco	MD
420	High	A	2.1%	USA	San Francisco	CA
333	High	B	3.0%	USA	San Francisco	CA
576	High	B	3.0%	USA	San Francisco	CA
128	Low	C	4.5%	UK	Reading	Berkshire
193	Low	C	4.5%	UK	London	London
550	Low	B	3.5%	UK	London	London

F1: [Income, OtherProd] → [Rate] F2: [Country, City] → [State]

How to prove it in the general case?

29

Closure Test for FDs

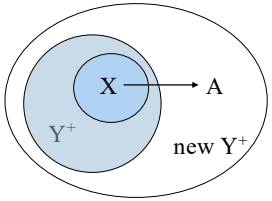
30

- Given attribute set Y and FD set F
  - Denote Y_F^+ or Y^+ the closure of Y relative to F
    Y_F^+ = set of all FDs given or implied by Y
- Computing the closure of Y
  - Start: $Y_F^+ = Y$, $F' = F$
  - While there exists an $f \in F'$ s.t. LHS(f) $\subseteq Y_F^+$:
    $Y^+_F = Y^+_F \cup RHS(f)$
    $F' = F' - f$
  - At end: $Y \rightarrow B$ for all $B \in Y_F^+$

Acknowledgements: M. Papagelis

30

Computing the closure Y^+ of a set of attributes Y
Given FDs F:

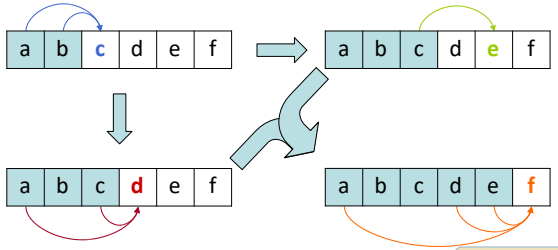


31

Example: Closure Test

32

- Consider R(a,b,c,d,e,f)
  with FDs ab → c, ac → d, c → e, ade → f
- Find Y^+ if Y = ab or find $\{a,b\}^+$



$\{a,b\}^+=\{a,b,c,d,e,f\}$ or $ab \rightarrow cdef$

ab is a candidate key!

32

Lecture Example

33

33

Lecture Example

34

34

Discarding redundant FDs

35

- Minimal basis: opposite from closure
- Given a set of FDs F, want a minimal F' s.t.
  - F' $\subseteq$ F
  - F' entails f for all f $\in$ F
- Properties of a minimal basis F'
  - RHS is always singleton
  - If any FD is removed from F', F' is no longer a minimal basis
  - If for any FD in F' we remove one or more attributes from the LHS of f $\in$ F', the result is no longer a minimal basis

35

In other words …

36

- *Minimal basis also referred to as minimal cover*
- *Minimal basis* for FDs:
  - Right sides are single attributes.
  - No FD can be removed.
  - No attribute can be removed from a left side.
- Constructing a minimal cover
  - Decompose RHS to single attributes
  - Repeatedly try to remove an FD and see if remaining FDs are equivalent to original set. That is, does the closure of the LHS attributes (of the removed FD) include the RHS attribute?
  - Repeatedly try to remove an attribute from a LHS and see if the removed attribute can be derived from the remaining FDs.

36

Constructing a minimal basis

37

Straightforward but time-consuming

1. Split all RHS into singletons
2. For all f in F', test whether J = $(F'-f)^+$ is still equivalent to F^+

=> Might make F' too small

3. For all i ∈ LHS(f), for all f∈ F', let LHS(f')=LHS(f)-i
   Test whether $(F'-f+f')^+$ is still equivalent to F^+
4. Repeat (2) and (3) until neither makes progress

37

Minimal Basis: Example

38

- Relation R: R(A, B, C, D)
- Defined FDs:
  - F = {A->AC, B->ABC, D->ABC}

Find the minimal basis M of F

38

Minimal Basis: Example (cont.)

39

F = {A->AC, B->ABC, D->ABC}

1st Step

- H = {A->A, A->C, B->A, B->B, B->C, D->A, D->B, D->C}

2nd Step

- A->A, B->B: can be removed as trivial
- A->C: can't be removed, as there is no other LHS with A
- B->A: can't be removed, because for J=H-{B->A} is B^+=BC
- B->C: can be removed, because for J=H-{B->C} is B^+=ABC
- D->A: can be removed, because for J=H-{D->A} is D^+=DBA
- D->B: can't be removed, because for J=H-{D->B} is D^+=DC
- D->C: can be removed, because for J=H-{D->C} is D^+=DBAC

Step outcome => H = {A->C, B->A, D->B}

39

Minimal Basis: Example (cont.)

40

3rd Step

- H doesn't change as all LHS in H are single attributes

4th Step

- H doesn't change

Minimal Basis: M = H = {A->C, B->A, D->B}

40

Lecture Example

41

41

Lecture Example

42

42

A Geometric View of FDs

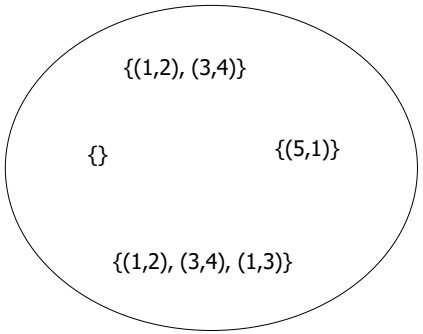
43

- Imagine the set of all *instances* of a particular relation.
- That is, all finite sets of tuples that have the proper number of components.
- Each instance is a point in this space.

43

Example: R(A,B)

44



44

An FD is a Subset of Instances

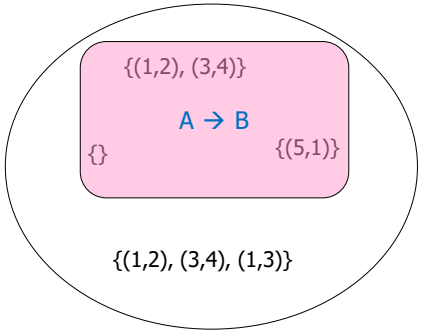
45

- For each FD $X \rightarrow A$ there is a subset of all instances that satisfy the FD.
- We can represent an FD by a region in the space.

45

Example: A → B for R(A,B)

46



46

Representing *Sets* of FDs

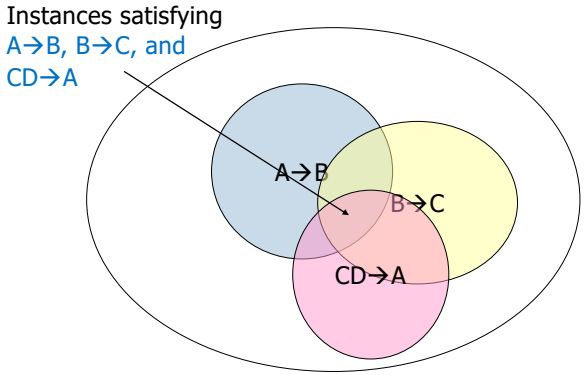
47

- If each FD is a set of relation instances, then a collection of FDs corresponds to the intersection of those sets.
  - Intersection = all instances that satisfy all of the FDs.

47

Example

48



48

Implication of FDs

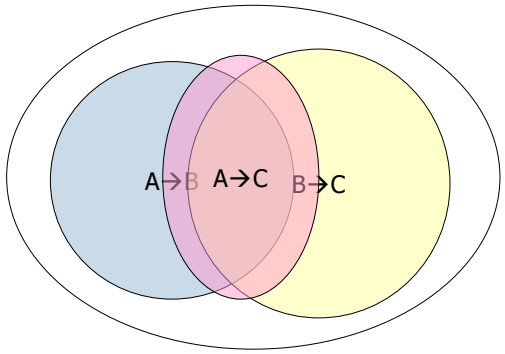
49

- If an FD $Y \rightarrow B$ follows from FDs $X_1 \rightarrow A_1,\ldots,X_n \rightarrow A_n$, then the region in the space of instances for $Y \rightarrow B$ must include the intersection of the regions for the FDs $X_i \rightarrow A_i$.
  - That is, every instance satisfying all the FDs $X_i \rightarrow A_i$ surely satisfies $Y \rightarrow B$.
  - But an instance could satisfy $Y \rightarrow B$, yet not be in this intersection.
- For a set of FDs ***F***, F^+ (the closure of ***F***) is the set of all FDs implied by ***F***

49

Example

50



50

Closure of F

51

- For a set of FDs ***F***, F^+ (the closure of ***F***) is the set of all FDs that can be derived (implied) from ***F***
  - Do not confuse closure of F with closure of an attribute set

51

Closure of F

52

- Example: Assume R(A, B, C, D), with F = {A→ B, B→C). Then F^+ includes the following FDs:

$A \rightarrow A, A \rightarrow B, A \rightarrow C, B \rightarrow B, B \rightarrow C, C \rightarrow C, D \rightarrow D,$
$AB \rightarrow A, AB \rightarrow B, AB \rightarrow C, AC \rightarrow A, AC \rightarrow B, AC \rightarrow C,$
$AD \rightarrow A, AD \rightarrow B, AD \rightarrow C, AD \rightarrow D, BC \rightarrow B, BC \rightarrow C,$
$BD \rightarrow B, BD \rightarrow C, BD \rightarrow D, CD \rightarrow C, CD \rightarrow D,$
$ABC \rightarrow A, ABC \rightarrow B, ABC \rightarrow C, ABD \rightarrow A, ABD \rightarrow B,$
$ABD \rightarrow C, ABD \rightarrow D, BCD \rightarrow B, BCD \rightarrow C, BCD \rightarrow D,$
$ABCD \rightarrow A, ABCD \rightarrow B, ABCD \rightarrow C, ABCD \rightarrow D.$

52

53

Part II:
Schema Decomposition

53

Relational Schema Design

54

- Goal of relational schema design is to avoid redundancy, and the anomalies it enables.
  - *Update anomaly* : one occurrence of a fact is changed, but not all occurrences have been updated.
  - *Deletion anomaly* : valid fact is lost when a tuple is deleted.

54

Result of bad design: Anomalies

55

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	Voyager	WickedAle	Pete's	WickedAle
Spock	Enterprise	Bud	A.B.	Bud

- Update anomaly: if Janeway is transferred to *Intrepid*, will we remember to change each of her tuples?
- Deletion anomaly: If nobody likes Bud, we lose track of the fact that Anheuser-Busch manufactures Bud.

55

Example of Bad Design

56

Suppose we have FDs name -> addr, favBeer and beersLiked -> manf. This design is bad:

Drinkers(name, addr, beersLiked, manf, favBeer)

name	addr	beersLiked	manf	favBeer
Janeway	Voyager	Bud	A.B.	WickedAle
Janeway	???	WickedAle	Pete's	???
Spock	Enterprise	Bud	???	Bud

Data is redundant, because each of the ???'s can be figured out by using the FDs.

56

Goal of Decomposition

57

- Eliminate redundancy by decomposing a relation into several relations
- Check that a decomposition does not lead to bad design

57

FDs and redundancy

58

Given relation R and FDs F

- R often exhibits anomalies due to redundancy
- F identifies many (not all) of the underlying problems

Idea

- Use F to identify "good" ways to split relations
- Split R into 2+ smaller relations having less redundancy
- Split F into subsets which apply to the new relations (compute the projection of functional dependencies)

58

Schema decomposition

59

- Given relation R and FDs F
  - Split R into R_i s.t. for all i $R_i \subset R$ (no new attributes)
  - Split F into F_i s.t. for all i, F entails F_i (no new FDs)
  - F_i involves only attributes in R_i
- Caveat: entirely possible to lose information
  - F^+ may entail FD f which is not in $(\cup_i F_i)^+$

  => Decomposition lost some FDs
  - Possible to have $R \subset \bowtie_i R_i$

  => Decomposition lost some relationships
- Goal: minimize anomalies without losing info

59

Good Properties of Decomposition

60

1) Lossless Join Decomposition
   - When we join decomposed relations we should get ***exactly*** what we started with
2) Avoid anomalies
   - Avoid redundant data
3) Dependency Preservation
   - $(F_1 \cup \ldots \cup F_n)^+ = F^+$

60

Problem with Decomposition

61

Given instances of the decomposed relations, we may not be able to reconstruct the corresponding instance of the original relation – information loss

61

Example: Splitting Relations

62

Student_Name	Student_Email	Course	Instructor
Alice	alice@gmail	SE 4DB3	Chiang
Alice	alice@gmail	CS 3SH3	Zheng
Bob	bob@mcmaster	SE 3RA3	Janicki
Laura	laura@gmail	SE 4DB3	Jones

Students (email, name)

Courses (code, instructor)

Taking (email, courseCode)

Students ⋈ Taking ⋈ Courses has additional tuples!

- (Alice, alice@gmail, SE4DB3, Jones), but Alice is not in Jones' section of SE 4DB3
- (Laura, laura@gmail, SE4DB3, Chiang), but Laura is not in Chiang's section of SE 4DB3

Why did this happen? How to prevent it?

62

Information loss with decomposition

63

- Decompose R into S and T
  - Consider FD A->B, with A only in S and B only in T
- FD loss
  - Attributes A and B no longer in same relation
  => Must join T and S to enforce A->B (expensive)
- Join loss
  - Neither (S ∩ T) -> S nor (S ∩ T) -> T in F^+
  => Joining T and S produces extraneous tuples

63

Lossless Join Decomposition

64

- A decomposition should not lose information
- A decomposition ($\mathbf{R}_1,\ldots,\mathbf{R}_n$) of a schema, **R**, is **lossless** if every valid instance, r, of **R** can be reconstructed from its components:
  - $r = r_1 \bowtie \ldots \bowtie r_n$ where $r_i = \Pi_{Ri}(r)$

64

Lossy Decomposition

65

r

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak

$r_1 = \Pi_{R1}(r)$

ID	Name
11	Pat
12	Jen
13	Jen

$r_2 = \Pi_{R2}(r)$

Name	Addr
Pat	1 Main
Jen	2 Pine
Jen	3 Oak

$r_1 \bowtie r_2$

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak
12	Jen	3 Oak
13	Jen	2 Pine

65

What is lost?

66

- Lossy decomposition
  - Loses the fact that (12, Jen) lives at 2 Pine (not 3 Oak)
  - Loses the fact that (13, Jen) lives at 3 Oak
- Remember: lossy decompositions yield ***more*** tuples than they should when relations are joined together

r

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak

$r_1 = \Pi_{R1}(r)$

ID	Name
11	Pat
12	Jen
13	Jen

$r_2 = \Pi_{R2}(r)$

Name	Addr
Pat	1 Main
Jen	2 Pine
Jen	3 Oak

ID	Name	Addr
11	Pat	1 Main
12	Jen	2 Pine
13	Jen	3 Oak
12	Jen	3 Oak
13	Jen	2 Pine

66

Example 2

67

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

R1

Model Name	Category
a11	Canon
s20	Nikon
a70	Canon

R2

Price	Category
100	Canon
200	Nikon
150	Canon

Ack: S.M. Lee

67

Example 2 (cont'd)

68

R1 ⋈ R2

Model Name	Price	Category
a11	100	Canon
a11	150	Canon
s20	200	Nikon
a70	100	Canon
a70	150	Canon

R

Model Name	Price	Category
a11	100	Canon
s20	200	Nikon
a70	150	Canon

68

Lossy decomposition

69

- Additional tuples are obtained along with original tuples
- Although there are more tuples, this leads to less information
- Due to the loss of information, the decomposition for the previous example is called lossy decomposition or lossy-join decomposition

69

Testing for Losslessness

70

- A (binary) decomposition of **R** = (R, **F**) into **R**1= (R1, **F**1) and **R**2 = (R2, **F**2) is lossless if and only if:
  - either the FD (R1 ∩ R2) → R1 is in **F**+
  - or the FD (R1 ∩ R2) → R2 is in **F**+
  - all attributes common to both R1 and R2 functionally determine ALL the attributes in R1 OR
  - all attributes common to both R1 and R2 functionally determine ALL the attributes in R2

70

Decomposition Property

71

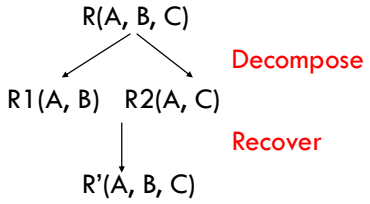
- In our example
  - *Name* ↛ ID,*Name*
  - *Name* ↛ *Name, Addr*
- *A lossless decomposition*
  - *[ID,Name] and [ID,Addr]*
- Example 2:
  - *Category* ↛ ModelName, Category
  - *Category* ↛ Price, Category
  - Better to use [MN, Category] and [MN, Price]
- In other words, if R1 ∩ R2 forms a superkey of either R1 or R2, the decomposition of R is a lossless decomposition

71

Lossless Decomposition

72

A decomposition is lossless if we can recover:



Thus, R' = R

72

Example : Lossless Decomposition

73

Given:

Lending-schema = (*branch-name, branch-city, assets, customer-name, loan-number, amount*)

FDs:

branch-name ⟶ *branch-city, assets*

loan-number ⟶ *amount, branch-name*

Decompose Lending-schema into two schemas:

Branch-schema = (*branch-name, branch-city, assets*)

Loan-info-schema = (*branch-name, customer-name, loan-number, amount*)

73

Example : Lossless Decomposition

74

Show that the decomposition is a Lossless Decomposition

Branch-schema = (*branch-name, branch-city, assets*)

Loan-info-schema = (*branch-name, customer-name, loan-number, amount*)

- Since *Branch-schema* ∩ *Loan-info-schema* = {*branch-name*}
- We are given: *branch-name* ⟶ *branch-city, assets*

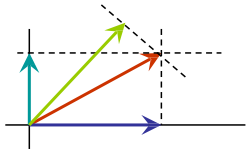
Thus, this decomposition is lossless.

74

Projecting FDs

75

- Once we've split a relation, we have to re-factor our FDs to match
  - Each FDs must only mention attributes from one relation
- Similar to geometric projection
  - Many possible projections (depends on how we slice it)
  - Keep only the ones we need (minimal basis)



75

Projecting FDs

76

- Given:
  - a relation *R*
  - the set *F* of FDs that hold in *R*
  - a relation $R_i \subset R$
- Determine the set of all FDs F_i that
  - Follow from *F* and
  - Involve only attributes of R_i

76

FD Projection Algorithm

77

- Start with $F_i = \emptyset$
- For each subset X of R_i
  - Compute X^+
  - For each attribute A in X^+
    - If A is in R_i
      - add X -> A to F_i
- Compute the minimal basis of F_i

77

Making projection more efficient

78

- Ignore trivial dependencies
  - No need to add *X* -> *A* if *A* is in *X* itself
- Ignore trivial subsets
  - The empty set or the set of all attributes (both are subsets of X)
- Ignore supersets of X if $X^+ = R$
  - They can only give us "weaker" FDs (with more on the LHS)

78

Example: Projecting FDs

79

- *Given R(A,B,C)* with FDs *A->B* and *B->C*
  - $A^+ = ABC$; yields *A->B, A->C*
    - We ignore A->A as trivial
    - We ignore the supersets of A, AB^+ and AC^+, because they can only give us "weaker" FDs (with more on the LHS)
  - $B^+ = BC$; yields *B->C*
  - $C^+ = C$; yields nothing.

79

Example cont'd

80

- Resulting FDs: *A->B, A->C*, and *B->C*
- Projection onto *AC* : *A->C*
  - Only FD that involves a subset of {*A*,*C*}
- Projection on BC: *B->C*
  - Only FD that involves subset of {B, C}

80

Projection is expensive

81

- Even with these tricks, projection is still expensive.
- Suppose R_1 has n attributes.
  How many subsets of R_1 are there?

$2^n - 1$

81

83

Part III: Normal Forms

83

Database Design Theory

84

- General idea:
  - Express constraints on the data
  - Use these to decompose the relations
- Ultimately, get a schema that is in a "normal form" that guarantees good properties, such as no anomalies.
- "Normal" in the sense of conforming to a standard.
- The process of converting a schema to a normal form is called normalization.

Acknowledgements: M. Papagelis, R. Johnson

84

Motivation for normal forms

85

- Identify a "good" schema
  - For some definition of "good"
  - Avoid anomalies, redundancy, etc.
- Many normal forms
  - 1st
  - 2nd
  - 3rd
  - Boyce-Codd
  - ... and several more we won't discuss…

$BCNF \subseteq 3NF \subseteq 2NF \subseteq 1NF$

85

1st Normal Form (1NF)

86

- No multi-valued attributes allowed
  - Imagine storing a list of values in an attribute
- Counter example
  - Course(name, instructor, [student,email]*)

Name	Instructor	Student Name	Student Email
CS 3DB3	Chiang	Alice	alice@gmail
		Mary	mary@mac
		Mary	mary@mac
SE 3SH3	Miller	Nilesh	nilesh@gmail

86

2nd normal form (2NF)

87

- Non-key attributes depend on candidate keys
  - Consider non-key attribute A
  - Then there exists an FD X s.t. X -> A, and X is a candidate key
- Counter-example
  - Movies(title, year, star, studio, studioAddress, salary)
  - FD: title, year -> studio; studio -> studioAddress; star->salary

Title	Year	Star	Studio	StudioAddr	Salary
Star Wars	1977	Hamill	Lucasfilm	1 Lucas Way	$100,000
Star Wars	1977	Ford	Lucasfilm	1 Lucas Way	$100,000
Star Wars	1977	Fisher	Lucasfilm	1 Lucas Way	$100,000
Patriot Games	1992	Ford	Paramount	Cloud 9	$2,000,000
Last Crusade	1989	Ford	Lucasfilm	1 Lucas Way	$1,000,000

87

3rd normal form (3NF)

88

- Non-prime attr. depend *only* on candidate keys
  - Consider FD X -> A
  - Either X is a superkey OR A is *prime* (part of a key)
- Counter-example:
  - studio -> studioAddr

  (*studioAddr* depends on *studio* which is not a candidate key)

Title	Year	Studio	StudioAddr
Star Wars	1977	Lucasfilm	1 Lucas Way
Patriot Games	1992	Paramount	Cloud 9
Last Crusade	1989	Lucasfilm	1 Lucas Way

88

3NF, dependencies, and join loss

89

- Theorem: always possible to convert a schema to lossless join, dependency-preserving 3NF
- Caveat: always *possible* to create schemas in 3NF for which these properties do not hold
- FD loss example 1:
  - MovieInfo(title, year, studioName)
  - StudioAddress(title, year, studioAddress)

  => Cannot enforce studioName -> studioAddress
- Join loss example 2:
  - Movies(title, year, star)
  - StarSalary(star, salary)

  => Movies ⋈ StarSalary yields additional tuples

89

Boyce-Codd normal form (BCNF)

90

- One additional restriction over 3NF
  - All non-trivial FDs have superkey LHS
- Counterexample
  - CanadianAddress(street, city, province, postalCode)
  - Candidate keys: {street, postalCode}, {street, city, province}
  - FD: postalCode -> city, province
  - Satisfies 3NF: city, province both prime
  - Violates BCNF: postalCode is not a superkey

  => Possible anomalies involving postalCode

90

Boyce-Codd Normal Form

91

- We say a relation R is in *BCNF* if whenever $X \rightarrow A$ is a nontrivial FD that holds in R, X is a superkey.
  - Remember: *non-trivial* means A is not contained in X.

91

Example: a relation not in BCNF

92

Drinkers(name, addr, beersLiked, manf, favBeer)

FD's: name→addr, favBeer, beersLiked→manf

- Only key is {name, beersLiked}.
- In each FD, the left side is **not** a superkey.
- Any one of these FDs shows *Drinkers* is not in BCNF

92

Another Example

93

Beers(name, manf, manfAddr)

FD's: name->manf, manf->manfAddr

- Beers w.r.t. name->manf does not violate BCNF, but manf->manfAddr does.

In other words, BCNF requires that:
the only FDs that hold are the result of key(s).
Why does that help?

93

Decomposition into BCNF

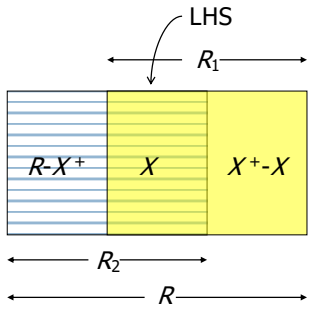
94

- Given: relation R with FDs F
- Look among the given FDs for a BCNF violation $X \rightarrow Y$ (i.e., X is not a superkey)
- Compute X^+.
  - Find $X^+ \neq X \neq$ all attributes, (o.w. X is a superkey)
- Replace R by relations with:
  - $R_1 = X^+$.
  - $R_2 = R - (X^+ - X) = R - X^+ \cup X$
- Continue to recursively decompose the two new relations
- *Project* given FDs F onto the two new relations.

94

Decomposition on X →Y

95



95

Example: BCNF Decomposition

96

Drinkers(name, addr, beersLiked, manf, favBeer)

F = {name→addr, name → favBeer, beersLiked→manf}

Key = name, beersLiked

- Pick BCNF violation name→addr.
- Closure: {name}$^+$ = {name, addr, favBeer}.
- Decomposed relations:
  - Drinkers1(name, addr, favBeer)
  - Drinkers2(name, beersLiked, manf)

96

Example -- Continued

97

- We are not done; we need to check Drinkers1 and Drinkers2 for BCNF.
- Projecting FDs is easy here.
- For Drinkers1(name, addr, favBeer), relevant FDs are name→addr and name→favBeer.
  - Thus, {name} is the only key and Drinkers1 is in BCNF.

97

Example -- Continued

98

- For Drinkers2(name, beersLiked, manf), the only FD is beersLiked→manf, and the only key is {name, beersLiked}.
  - Violation of BCNF.
- beersLiked$^+$ = {beersLiked, manf}, so we decompose *Drinkers2* into:
  - Drinkers3(beersLiked, manf)
  - Drinkers4(name, beersLiked)

98

Example -- Concluded

99

- The resulting decomposition of *Drinkers* :
  - Drinkers1(name, addr, favBeer)
  - Drinkers3(beersLiked, manf)
  - Drinkers4(name, beersLiked)
- Notice: *Drinkers1* tells us about drinkers, *Drinkers3* tells us about beers, and *Drinkers4* tells us the relationship between drinkers and the beers they like.

99

What we want from a decomposition

100

- *Lossless Join* : it should be possible to project the original relations onto the decomposed schema, and then reconstruct the original,
  i.e., get back exactly the original tuples.
- *No anomalies*
- *Dependency Preservation* : All the original FDs should be satisfied.

100

What we get from a BCNF decomposition

101

- *Lossless Join* : ✓
- *No anomalies* : ✓
- *Dependency Preservation* : ✗

101

Example: Failure to preserve dependencies

102

- Suppose we start with R(*A,B,C*) and FDs
  - *AB* →*C* and *C* →*B*.
- There are two keys, {*A,B* } and {*A,C* }.
- *C* →*B* is a BCNF violation, so we must decompose into *AC*, *BC*.

The problem is that if we use *AC* and *BC* as our database schema, we cannot enforce the FD *AB* →*C* in these decomposed relations.

102

3NF: Helps Us Avoid this Problem

103

- *3rd Normal Form* (3NF) modifies the BCNF condition so we do not have to decompose in this problem situation.
- An attribute is *prime* if it is a member of any key.
- $X \rightarrow A$ violates 3NF if and only if X is not a superkey, and also A is not prime.
- i.e., it's ok if X is not a superkey, as long as A is prime.

103

Example: 3NF

104

- In our problem situation with FDs $AB \rightarrow C$ and $C \rightarrow B$, we have keys AB and AC.
- Thus A, B, and C are each prime.
- Although $C \rightarrow B$ violates BCNF, it does not violate 3NF.

104

What we get from a 3NF decomposition

105

- *Lossless Join* : ✓
- *No anomalies* : ✗
- *Dependency Preservation* : ✓

Why?

Unfortunately, neither BCNF nor 3NF can guarantee all three properties we want.

105

3NF Synthesis Algorithm

106

- We can always construct a decomposition into 3NF relations with a lossless join and dependency preservation.
- Need *minimal basis* for the FDs (same as used in projection)
  - Right sides are single attributes.
  - No FD can be removed.
  - No attribute can be removed from a left side.

106

3NF Synthesis – (2)

107

- One relation for each FD in the minimal basis.
  - Schema is the union of the left and right sides.
- If no key is contained in an FD, then add one relation whose schema is some key.

107

Example: 3NF Synthesis

108

- Relation R = ABCD.
- FDs $A \rightarrow B$ and $A \rightarrow C$.
- Decomposition: AB and AC from the FDs, plus AD for a key.

108

Limits of decomposition

109

- Pick two…
  - Lossless join
  - Dependency preservation
  - Anomaly-free
- 3NF
  - Provides lossless join and dependency preserving
  - May allow some anomalies
- BCNF
  - Anomaly-free, lossless join
  - Sacrifice dependency preservation

Use domain knowledge to choose 3NF vs. BCNF

109

TRANSACTIONS & CONCURRENCY CONTROL

Adapted from K. Goldberg (UC Berkeley) and Ramakrishnan & Gherke

1

Introduction

2

- Concurrent execution of user programs is essential for good DBMS performance.
- Because disk accesses are frequent, and relatively slow, it is important to keep the CPU going by working on several user programs concurrently.

2

Transactions

3

- A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the database.
- A *transaction* is the DBMS's abstract view of a user program: a sequence of reads and writes.

3

Concurrency

4

- What is Concurrent Process (CP)?
  - Multiple users access databases and use computer systems simultaneously.
- Example: Airline reservation system.
  - An airline reservation system is used by hundreds of travel agents and reservation clerks concurrently.
  - Banking system: you may be updating your account balances the same time the bank is crediting you interest.

4

Concurrency

5

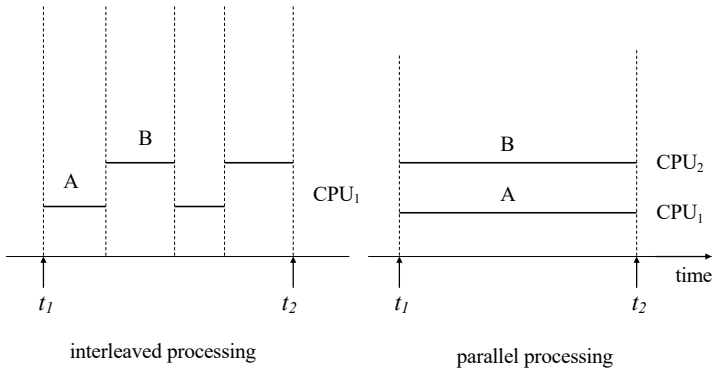
- Why Concurrent Process?
- Better transaction throughput and response time
- Better utilization of resource

- Concurrency
  - **Interleaved processing:**
    - Concurrent execution of processes is interleaved in a single CPU
  - **Parallel processing:**
    - Processes are concurrently executed in multiple CPUs.

5

Concurrent Transactions

6



6

Transactions

7

- What is a transaction?

  A sequence of many actions which are considered to be one unit of work.

  Basic operations a transaction can include "actions":
  - Reads, writes
  - Special actions: commit, abort

7

Concurrency (cont'd)

8

- Users submit transactions, and can think of each transaction as executing by itself.
  - Concurrency is achieved by the DBMS, which interleaves actions (reads/writes of DB objects) of various transactions.
  - Each transaction must leave the database in a consistent state if the DB is consistent when the transaction begins.
    - DBMS will enforce some constraints
    - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
- *Issues:* Effect of *interleaving* transactions, and *crashes.*

8

Atomicity of Transactions

9

- A transaction might *commit* after completing all its actions, or it could *abort* (or be aborted by the DBMS) after executing some actions.
- A very important property guaranteed by the DBMS for all transactions is that they are *atomic.* That is, a user can think of a Xact as always executing all its actions in one step, or not executing any actions at all.
  - DBMS *logs* all actions so that it can *undo* the actions of aborted transactions.

9

Transaction Properties (ACID)

10

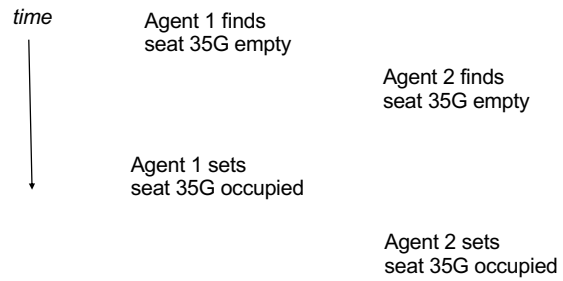
- Atomicity
  - Transaction is either performed in its entirety or not performed at all, this should be DBMS' responsibility
- Consistency
  - Transaction must take the database from one consistent state to another.
- Isolation
  - Transaction should appear as though it is being executed in isolation from other transactions
- Durability
  - Changes applied to the database by a committed transaction must persist, even if the system fails before all changes reflected on disk

10

Oops, Something's Wrong

11

- Reserving a seat for a flight
- In concurrent access to data in DBMS, two users may try to book the same seat simultaneously



11

Example

12

- Consider two transactions (*Xacts*):

```
T1:   BEGIN  A=A+100,   B=B-100   END
T2:   BEGIN  A=1.06*A,  B=1.06*B  END
```

- Intuitively, the first transaction is transferring $100 from B's account to A's account. The second is crediting both accounts with a 6% interest payment.
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect *must* be equivalent to these two transactions running serially in some order.

12

Example (Contd.)

13

- Consider a possible interleaving (*schedule*):

T1:	A=A+100,		B=B-100
T2:		A=1.06*A,	B=1.06*B

- This is OK. But what about:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

- The DBMS's view of the second schedule:

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	

13

What Can Go Wrong?

14

- Concurrent process may end up violating Isolation property of transaction if not carefully scheduled
- Transaction may be aborted before committed
  - undo the uncommitted transactions
  - undo transactions that see the uncommitted change before the crash

14

Schedule

15

- A schedule S of n transactions T1,T2,…Tn is an ordering of the operations of the transactions subject to the constraint that,
  - for each transaction Ti that participates in S, the operations of Ti in S must appear in the same order in which they occur in Ti.
- Informally, a schedule is a sequence of interleaved actions from all transactions
- Example:

S_a: R1(A),R2(A),W1(A),W2(A), Abort1,Commit2;

T1	T2
Read(A)	
	Read(A)
Write(A)	
	Write(A)
Abort T1	
	Commit T2

15

Scheduling Transactions

16

- *Serial schedule:* Schedule that does not interleave the actions of different transactions.
- *Equivalent schedules:* For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- *Serializable schedule:* A schedule that is equivalent to some serial execution of the transactions.

16

Serial Schedule

17

T1	T2
R(A)	
A := A+100	
R(B)	
B := B+100	
	R(A)
	A: = A* 2
	R(B)
	B:= B*2

S: R1(A),W1(A),R1(B), W1(B), R2(A), W2(A), R2(B), W2(B)

T1 T2

Adapted from M. Balazinska

17

A Serializable Schedule

18

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
R(B)	
B := B+100	
	R(B)
	B:= B*2

Notice: this is not a serial schedule, i.e., there is interleaving of operations

Net effect is the same as the serial schedule

S: R1(A),W1(A), R2(A), W2(A), R1(B), W1(B), R2(B), W2(B)

18

A Non-Serializable Schedule

19

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
	R(B)
	B:= B*2
R(B)	
B := B+100	

What's different?

S: R1(A),W1(A), R2(A), W2(A), R2(B), W2(B), R1(B), W1(B)

19

Conflict operations

20

- Two operations in a schedule are said to be *conflict* if they satisfy all three of the following conditions:
  (1) They belong to different transactions
  (2) They access the same item A;
  (3) At least one of the operations is a write(A)

Example in Sa: R1(A), R2(A), W1(A), W2(A), A1, C2;

- R1(A),W2(A) conflict, so do R2(A),W1(A),
- R1(A), W1(A) do not conflict because they belong to the same transaction,
- R1(A),R2(A) do not conflict because they are both read operations.

20

Assignment 3

16

- Posted on Avenue and MS Teams
- Due: December 3, 2021
- Recommend you start early, don't leave it until the last minute.

16

Scheduling Transactions

17

- *Serial schedule:* Schedule that does not interleave the actions of different transactions.
- *Equivalent schedules:* For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- *Serializable schedule:* A schedule that is equivalent to some serial execution of the transactions.

17

Serial Schedule

18

T1	T2
R(A)	
A := A+100	
R(B)	
B := B+100	
	R(A)
	A: = A* 2
	R(B)
	B:= B*2

S: R1(A),W1(A),R1(B), W1(B), R2(A), W2(A), R2(B), W2(B)

T1 T2

Adapted from M. Balazinska

18

A Serializable Schedule

19

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
R(B)	
B := B+100	
	R(B)
	B:= B*2

Notice: this is not a serial schedule, i.e., there is interleaving of operations

Net effect is the same as the serial schedule

S: R1(A),W1(A), R2(A), W2(A), R1(B), W1(B), R2(B), W2(B)

19

A Non-Serializable Schedule

20

T1	T2
R(A)	
A := A+100	
	R(A)
	A: = A* 2
	R(B)
	B:= B*2
R(B)	
B := B+100	

What's different?

Ordering of operations is not consistent

S: R1(A),W1(A), R2(A), W2(A), R2(B), W2(B), R1(B), W1(B)

20

Conflict operations

21

- Two operations in a schedule are said to be in *conflict* if they satisfy all three of the following conditions:
  (1) They belong to different transactions
  (2) They access the same item A;
  (3) At least one of the operations is a write(A)

Example in Sa: R1(A), R2(A), W1(A), W2(A), A1, C2;

- R1(A),W2(A) conflict, so do R2(A),W1(A),
- R1(A), W1(A) do not conflict because they belong to the same transaction,
- R1(A),R2(A) do not conflict because they are both read operations.

21

Conflicts

22

What can go wrong:

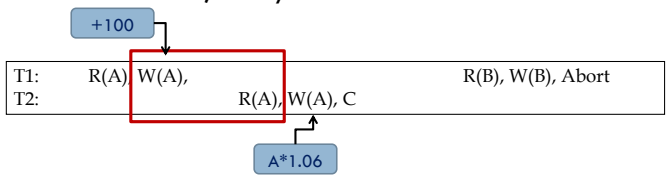
- Reading uncommitted data (WR), aka "dirty reads"
- Unrepeatable reads (RW)
- Lost updates (WW)

22

Reading Uncommitted Data

23

- WR Conflicts, "dirty reads"



- What's the problem?

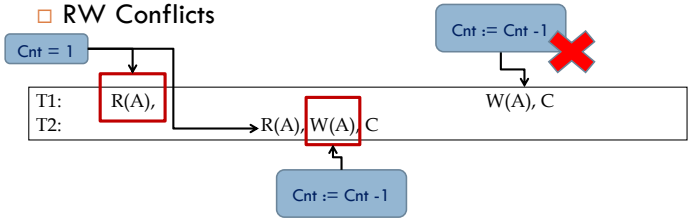
T2 has read an updated value A, which is later aborted!

23

Unrepeatable Reads

24

- RW Conflicts



- What's the problem?

T2 has changed A that has been read by T1, while T1 is still in progress

24

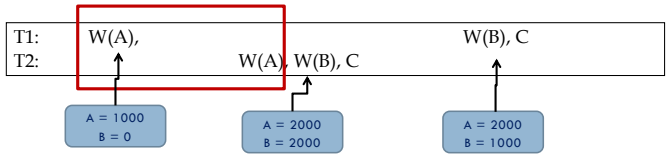
Lost Updates

25

T1: sets salaries to $1000
T2: sets salaries to $2000

Require: A and B salaries to be equal

- Overwriting uncommitted data
- WW Conflicts



- What's the problem?

T2 overwrites T1's update of A, while T1 is still in progress.

25

Schedules with Aborts

26

Withdraw $100 from A

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
Abort	

Bank reads A, B, and credits 6% interest

What now? (T2 read a value that is no longer valid)

26

Options

27

- If T2 did not commit, we abort T1 and cascade to T2.
- But T2 committed, so we cannot undo
- This schedule is *unrecoverable*

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	R(B)
	W(B)
	Commit
Abort	

27

Aborting a Transaction

28

- If a transaction *Ti* is aborted, all its actions have to be undone. Not only that, if *Tj* reads an object last written by *Ti*, *Tj* must be aborted as well!
- Most systems try to avoid such *cascading aborts*
  - If *Ti* writes an object, *Tj* can read this only after *Ti* commits.

28

Aborting a Transaction (cont'd)

29

- In order to *undo* the actions of an aborted transaction, the DBMS maintains a *log* in which every write is recorded.
- This mechanism is also used to recover from system crashes: all active Xacts at the time of the crash are aborted when the system comes back up.

29

Recoverable Schedules and Avoid Cascading Aborts

30

Recoverable

- Aborting T1 requires aborting T2
  - But T2 has already committed!
- A recoverable schedule is one in which this cannot happen.
  - i.e. a Xact commits only after all the Xacts it "depends on" (i.e. it reads from) commit.

Avoid cascading abort (ACA)

- Aborting T1 requires aborting T2!
- Aborting a Xact can be done without cascading the abort to other Xacts.
- A Xact only reads data from committed Xacts.
- ACA implies recoverable (but not vice-versa!)

No		Yes	
T1	T2	T1	T2
R(A)		R(A)	
W(A)		W(A)	
	R(A)		R (A)
	W(A)		W (A)
	Commit	Commit	
Abort			Commit

T1	T2	T1	T2
R (A)		R (A)	
W (A)		W (A)	
	R (A)	Commit	
Abort	W (A)		R (A)
			W (A)

30

Example

31

W1(X), R2(Y), R1(Y), R2(X), C2, C1

T1, T2: W1(X), R1(Y), R2(Y), R2(X)

- Serializable: Yes, equivalent to T1,T2
- Recoverable: No. Yes, if C1 and C2 are switched
- ACA: No.

  Yes, if T1 commits before T2 reads X.

31

Including Aborts in Serializability

32

- Extend the definition of a serializable schedule to include aborts
- Serializable schedule: a schedule that is equivalent to some serial execution of the set of *committed* transactions.

32

Conflict Serializable Schedules

33

- Two schedules are conflict equivalent if:
  - Involve the same actions of the same transactions
  - Every pair of conflicting actions is ordered the same way
- Schedule S is conflict serializable if S is conflict equivalent to some serial schedule

33

Recall Conflicts

34

- Two writes by Ti, Tj to same element
  - Wi(X); Wj(X)
- Read/write by Ti, Tj to same element
  - Wi(X); Rj(X)
  - Ri(X); Wj(X)

34

Conflict Equivalent

35

- Outcome of a schedule depends on the order of conflicting operations
- Can interchange non-conflicting ops without changing effect of the schedule
- If two schedules S1 and S2 are conflict equivalent then they have the same effect

  S1 ←→ S2 by swapping non-conflicting ops

35

Example Slide

36

36

Precedence Graph Test

37

Is a schedule conflict-serializable ?

Simple test:

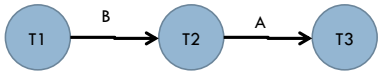
- Build a graph of all transactions Ti
- Edge from Ti to Tj if Ti comes first, and makes an action that conflicts with one of Tj
- The test: if the graph has no cycles, then it is conflict serializable !

37

Example 1

38

R2(A); R1(B); W2(A); R3(A); W1(B); W3(A); R2(B); W2(B)



This schedule is conflict serializable

38

Example Slide

39

39

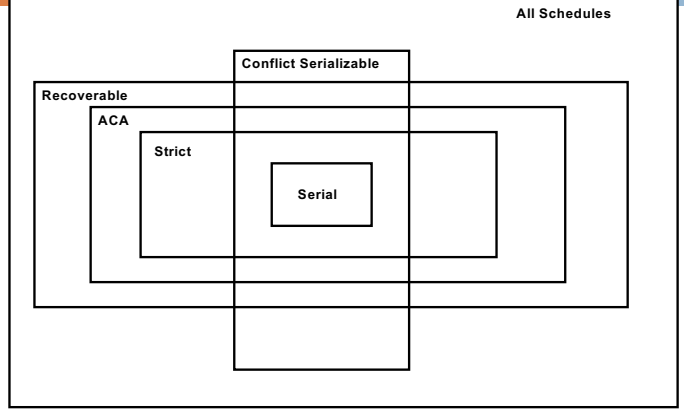
Strict Schedule

40

- A schedule S is strict if a value written by Ti is not read or overwritten by other Tj until Ti aborts or commits
- Example:
  W1(A); W1(B), C1; W2(A); R2(B); C2;
- Strict schedules are recoverable, and avoid cascading aborts.

40

Venn Diagram for Schedules



41

Scheduler

42

- The scheduler is the module that schedules the transaction's actions, ensuring serializability.
- How ?
  - Locks
  - Time stamps

42

Lock Based Concurrency Control

43

- DBMS aims to allow only recoverable and serializable schedules
- Ensure committed transactions are not un-done while aborting transactions
- Use a *locking protocol*: a set of rules to be followed by each transaction to ensure serializable schedules
- *Lock*: a mechanism to control concurrent access to a data object

43

Locking Scheduler

44

Simple idea:

- Each element has a unique lock
- Each transaction must first acquire the lock before reading/writing that element
- If the lock is taken by another transaction, then wait
- The transaction must release the lock(s)

Adapted from M. Balazinska

44

Notation

45

- Li(A) = transaction Ti acquires lock for element A
- Ui(A) = transaction Ti releases lock for element A

45

Example 1

46

T1	T2
L1(A)	
R1(A), W1(A)	
U1(A), L1(B)	
	L2(A)
	R2(A), W2(A)
	U2(A)
	L2(B) , **DENIED…**
R1(B), W1(B)	
U1(B)	
	GRANTED;
	R2(B), W2(B)
	U2(B)

Scheduler has enforced a conflict serializable schedule

46

Example 2

47

T1	T2
L1(A)	
R1(A), W1(A)	
U1(A)	
	L2(A)
	R2(A), W2(A)
	U2(A)
	L2(B)
	R2(B), W2(B)
	U2(B)
L1(B)	
R1(B), W1(B)	
U1(B)	

47

Types of Locks

48

- Shared lock (for reading)
- Exclusive lock (for writing, and of course, also for reading)
- Notation
  - $S_T(A)$: transaction T requests shared lock on object A
  - $X_T(A)$: transaction T requests exclusive lock on object A

48

Lock Modes

49

- S = Shared lock (for read)
- X = Exclusive lock (for write)

Lock compatibility matrix

	None	S	X
None	OK	OK	OK
S	OK	OK	Conflict
X	OK	Conflict	Conflict

49

Strict Two Phase Locking (Strict 2PL)

50

- Most widely used locking protocol
- Two rules:
  1. Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  2. All locks held by a transaction are released when the transaction completes
- If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.
- Strict 2PL allows only schedules whose precedence graph is acyclic (i.e., serializable)
- Recoverable and ACA

50

Strict 2PL Example

51

T1	T2
L(A);	
R(A), W(A)	
	L(A); **DENIED…**
L(B);	
R(B), W(B)	
U(A), U(B)	
Commit;	
	…GRANTED
	R(A), W(A)
	L(B);
	R(B), W(B)
	U(A), U(B)
	Commit;

All locks held by T1 are released when T1 completes.

51

Implications

52

- The locking protocol only allows safe interleavings of transactions
- If T1 and T2 access different data objects, then no conflict and each may proceed
- Otherwise, if same object, actions are ordered serially.
  - The Xact who gets the lock first must complete before the other can proceed

52

Two Phase Locking Protocol (2PL)

53

- Variant of Strict 2PL
- Relaxes the 2nd rule of Strict 2PL to allow Xacts to release locks before the end (commit/abort)
- Two rules:
  - Each Xact must obtain a S (*shared*) lock on object before reading, and an X (*exclusive*) lock on object before writing.
  - A transaction cannot request additional locks once it releases any lock.
  - If an Xact holds an X lock on an object, no other Xact can get a lock (S or X) on that object.

53

2PL Example

54

T1	T2
X(A), X(B)	
R(A), W(A)	
U(A)	
	X(A)
	R(A), W(A)
	X(B), **DENIED…**
R(B), W(B)	
U(B)	
	..GRANTED
	R(B), W(B)
	U(A), U(B)

All locks are first acquired, and then released.

54

2PL Implications

55

- In every transaction, all lock requests must precede all unlock requests.
- This ensures conflict serializability
  - Why? (Think of order Xacts enter their shrinking phase)
  - This induces a sort ordering of the transactions that can be serialized

55

Strict 2PL makes 2PL "strict"

56

- Recall: a strict schedule is one where a value written by T is not read/overwritten until T commits/aborts
  - Strict 2PL makes T hold locks until commit/abort
  - No other transaction can see or modify the data object until T is complete

56

Remarks

57

- What if a transaction releases its locks and then aborts?
- Recall: conflict serializable definition only considers *committed* transactions

57

Phantom Problem

58

- So far we have assumed the database to be a *static* collection of elements (=tuples)
- If tuples are inserted/deleted then the *phantom problem* appears

58

Phantom Problem

59

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('gizmo','blue')
SELECT * FROM Product WHERE color='blue'	

Is this schedule serializable ?

59

Phantom Problem

60

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('gizmo','blue')
SELECT * FROM Product WHERE color='blue'	

Suppose there are two blue products, X1, X2:

R1(X1),R1(X2),W2(X3),R1(X1),R1(X2),R1(X3)

This is conflict serializable ! What's wrong ??

60

Phantom Problem

61

T1	T2
SELECT * FROM Product WHERE color='blue'	
	INSERT INTO Product(name, color) VALUES ('gizmo','blue')
SELECT * FROM Product WHERE color='blue'	

Suppose there are two blue products, X1, X2:

R1(X1),R1(X2),W2(X3),R1(X1),R1(X2),R1(X3)

61

Phantom Problem

62

- A "phantom" is a tuple that is invisible during part of a transaction execution but not all of it.
- In our example:
  - T1: reads list of products
  - T2: inserts a new product
  - T1: re-reads: a new product appears !

62

Phantom Problem

63

- In a ***static*** database:
  - Conflict serializability implies serializability
- In a ***dynamic*** database, this may fail due to phantoms

63

Dealing With Phantoms

64

- Lock the entire table, or
- Lock the index entry for 'blue'
  - If index is available

Dealing with phantoms is expensive !

64

Transaction Support in SQL

65

- A transaction is initiated implicitly when an SQL statement is executed.
  - Each DBMS provides either a commit or a rollback (abort) option
- Isolation level: controls the extent to which a transaction is exposed to actions of other transactions executing concurrently
- Four possible isolation levels
  - Increase concurrency → increasing Xact exposure to uncommitted changes from other Xacts

65

Isolation Levels (cont'd)

66

- SQL-92 provides different isolation levels that control the degree of concurrency

Isolation Level	In DB2	Dirty Read	Unrepeatable Read	
Read Uncommitted	Uncommitted Read	Maybe	Maybe	Phantoms possible
Read Committed	Cursor Stability (default)	No	Maybe	Phantoms possible
Repeatable Reads	Repeatable Reads	No	No	Phantoms possible
Serializable	Read Stability	No	No	

66

Degrees of Isolation in SQL

67

- **Four levels of isolation**
  - READ UNCOMMITTED: no read locks
  - READ COMMITTED: short duration read locks
  - REPEATABLE READ:
    - Long duration read locks on individual items
  - SERIALIZABLE:
    - All locks long duration
- **Trade-off: consistency vs concurrency**
- Commercial systems give choice of level

67

Isolation Levels in SQL

68

1. "Dirty reads"
   SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
2. "Committed reads"
   SET TRANSACTION ISOLATION LEVEL READ COMMITTED
3. "Repeatable reads"
   SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
4. Serializable transactions
   SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

ACID

68

Choosing Isolation Level

69

- Trade-off: efficiency vs correctness
- DBMSs give user choice of level

Beware!!
• Default level is often NOT serializable
• Default level differs between DBMSs
• Serializable may not be exactly ACID

Always read docs!

69

Performance of Locking

70

- Locking aims to resolve conflicts among transactions by:
  - Blocking
  - Aborting

  } Performance penalty

- Blocked Xacts hold locks other Xacts may want
- Aborting Xact wastes work done thus far
- Deadlock: Xact is blocked indefinitely until one of the Xacts is aborted

70

Locking Performance

71

- Locking performance problems are *common!*
- The problem is too much blocking.
- The solution is to reduce the "locking load"
- Good heuristic – If more than 30% of transactions are blocked, then reduce the number of concurrent transactions

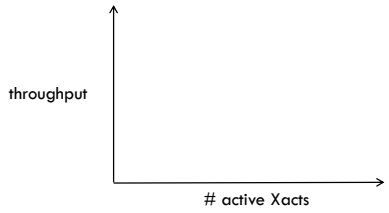
Credit: Phil Bernstein

71

Performance

72

- Locking overhead is primarily due to delays from blocking; minimizes throughout
- What happens to throughput as you increase the # of Xacts?



72

Improving Performance

73

- Lock the smallest sized object
  - Reduce likelihood that two Xacts need the same lock
- Reduce the time Xacts hold locks
- Reduce hot spots. A hot spot is an object that is frequently accessed, and causes blocking delays

73

Locking Granularity

74

- Granularity - size of data items to lock
  - e.g., files, pages, records, fields
- Coarse granularity implies
  - very few locks, so little locking overhead
  - must lock large chunks of data, so high chance of conflict, so concurrency may be low
- Fine granularity implies
  - many locks, so high locking overhead
  - locking conflict occurs only when two transactions try to access the exact same data concurrently

74

Deadlocks

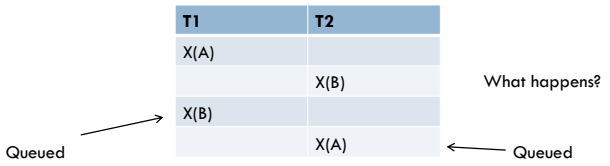
75

- Deadlock: Cycle of transactions waiting for locks to be released by each other.
- Two ways of dealing with deadlocks:
  - Deadlock detection
  - Deadlock prevention

75

Deadlocks

76



T1 is waiting for T2 to release its lock
T2 is waiting for T1 to release its lock
→ Such a cycle of transactions is a *deadlock*

Implications:
•T1 and T2 will make no further progress
•They may hold locks needed by other Xacts
•DBMS tries to prevent or detect (and resolve) deadlocks

76

Deadlock Detection

77

- Detect deadlocks automatically, and abort a deadlocked transaction (the *victim*).
- Preferred approach, because it allows higher resource utilization
- Timeout-based deadlock detection - If a transaction is blocked for too long, then abort it.
  - Simple and easy to implement
  - But aborts unnecessarily (pessimistic) and
  - some deadlocks persist for too long

77

Deadlock Detection

78

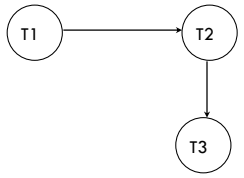
- Create a waits-for graph:
  - Nodes are transactions
  - There is an edge from Ti to Tj if Ti is waiting for Tj to release a lock
  - Lock mgr adds edge when lock request is queued
  - Remove edge when lock request granted
- A deadlock exists if there is a cycle in the waits-for graph
- Periodically check for cycles

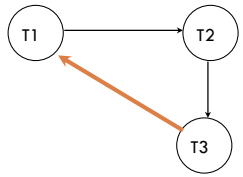
78

Waits-For Graph: Example

79

T1: S(A), R(A), S(B)
T2: X(B),W(B) X(C)
T3: S(C), R(C)





79

Lecture Example

80

80

Selecting a Victim

81

- Deadlock is resolved by aborting a Xact in the cycle, and releasing its locks
- Different criteria may be used:

81

Commercial DBMS Approaches

82

- **MS SQL Server**: Aborts the transaction that is "cheapest" to roll back.
  - "Cheapest" is determined by the amount of log generated.
  - Allows transactions that you've invested a lot in, to complete.
- **Oracle:** The transaction that detects the deadlock is the victim.
- **DB2:** the deadlock detector arbitrarily selects one deadlocked process as the victim to roll back.

82

Deadlock Prevention

83

- When there is a high level of lock contention and an increased likelihood of deadlocks
- Prevent deadlocks by giving each Xact a priority
- Assign priorities based on timestamps.
  - Lower timestamp indicates higher priority
  - i.e., oldest transaction has the highest priority
  - Higher priority Xacts cannot wait for lower priority Xacts (or vice versa)

83

Deadlock Prevention (cont'd)

84

- Assume Ti wants a lock that Tj holds. Two policies are possible:
  - **Wait-Die:** If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts
  - **Wound-wait:** If Ti has higher priority, Tj aborts; otherwise Ti waits
- Both schemes will cause aborts even though deadlock may not have occurred.
- If a transaction re-starts, make sure it has its original timestamp – WHY?

84

Wait-Die

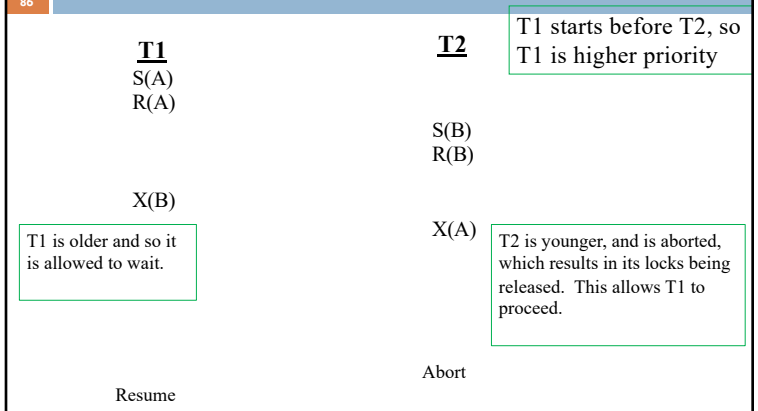
85

- Suppose Ti tries to lock an item already locked by Tj.
- If Ti is the older transaction then Ti will wait
- Otherwise Ti is aborted and re-starts later with the same timestamp.
- Lower priority transactions never wait for higher priority transactions.

85

Example: Wait-Die

86



86

Wound-Wait

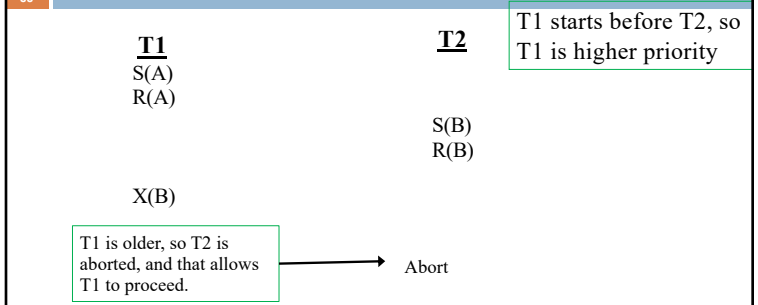
87

- Suppose Ti tries to lock an item locked by Tj.
- If Ti is higher priority (older transaction)
  - then Tj is aborted and restarts later with the same timestamp;
- Otherwise, Ti waits.

87

Example: Wound-Wait

88



88

101

Lock Conversions

101

Lock Conversions

102

- Lock upgrade: Xact that holds a shared lock can be upgraded to hold an exclusive lock
  - Reduces concurrency
- Lock downgrade: Xact that holds an exclusive lock, downgrades to a shared lock.
  - Improves concurrency (holding locks for writing when not required)
  - Reduces deadlocks

102

2PL with lock conversions

103

- During growing phase:
  - can acquire an S-lock on item
  - can acquire an X-lock on item
  - can convert an S-lock to an X-lock (upgrade)
  - Special case: allow lock downgrades only if Xact did not modify the data object (read only)
- During shrinking phase:
  - can release an S-lock
  - can release an X-lock
  - can convert an X-lock to an S-lock (downgrade)

Credit: Y. Chen, P. Bernstein, R. Ramakrishnan, J. Gehrke

103

Multiple Granularity Locking (MGL)

104

- We've referred to locking 'data objects'
- Sometimes preferable to group several data objects together
  - E.g., locking all records in a file
  - Define multiple levels of locking *granularity*
- Database consists of different data objects:
  - a field (attribute)
  - a database record
  - a page
  - a table
  - a file
  - the entire database

104

MGL (cont'd)

105

- Coarse grained locking → lower degree of concurrency
- Fine grained locking → higher degree of concurrency
  → Increased locking overhead
- What is the best locking granularity?

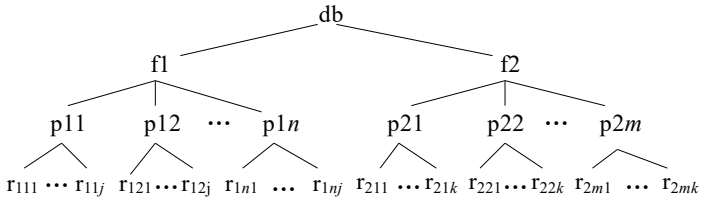
It depends on the transactions and the application

105

Granularity Hierarchy

106

- Most DBMS support multiple levels of locking granularity for different transactions



106

Problem with S and X Locks

107

T1: updates all the records in file f1.
T2: read record r_{1nj}.

Assume that T1 comes before T2:

- T1 locks f1.
- Before T2 is executed, the compatibility of the lock on r_{1nj} with the lock on f1 should be checked.

Assume that T2 comes before T1:

- T2 locks r_{1nj}.
- Before T1 is executed, the compatibility of the lock on f1 with the lock on r_{1nj} should be checked.
- Lock manager must efficiently manage all lock requests across hierarchy
- Each data object (e.g., file or record) has a different id

107

Solution

108

- Exploit the natural hierarchy of data containment
- Before locking fine-grained data, set *intention locks* on coarse grained data that contains it
- E.g., before setting an S-lock on a record, get an intention-shared-lock on the table that contains the record

108

Intention Locks

109

Three types of intention locks:

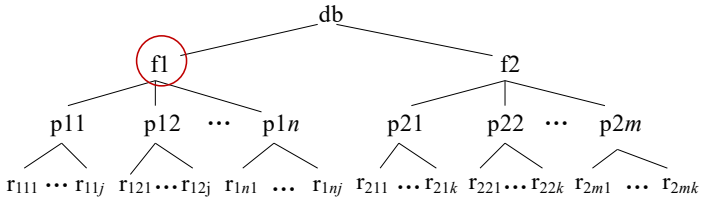
1. Intention-shared (IS) indicates that a shared lock(s) will be requested on some descendant node(s).
2. Intention-exclusive (IX) indicates that an exclusive lock(s) will be requested on some descendant node(s).
3. Shared-intention-exclusive (SIX) indicates that the current node is locked in shared mode but an exclusive lock(s) will be requested on some descendant node(s).

109

Back to our example

110

T1: updates all the records in file f1.
T2: read record r_{1nj}.



110

Compatibility Matrix

111

SIX = read with intent to write, e.g., for a scan that updates some of the records it reads

	IS	IX	S	SIX	X
IS	yes	yes	yes	yes	no
IX	yes	yes	no	no	no
S	yes	no	yes	no	no
SIX	yes	no	no	no	no
X	no	no	no	no	no

IS conflicts with X because IS says there's a fine-grained S-lock that conflicts with an X-lock

111

Multiple Granularity Lock Protocol

112

- Each Xact starts from the root of the hierarchy.
- To get S or IS lock on a node, must hold IS or IX on parent node.
- To get X or IX or SIX on a node, must hold IX or SIX on parent node.
- Must release locks in bottom-up (leaf to root) order.

112

Examples

113

- T1 scans R, and updates a few tuples:
  - T1 gets an SIX lock on R, then repeatedly gets an S lock on tuples of R, and occasionally upgrades to X on the tuples.
- T2 uses an index to read only part of R:
  - T2 gets an IS lock on R, and repeatedly gets an S lock on tuples of R.
- T3 reads all of R:
  - T3 gets an S lock on R.
  - OR, T3 could behave like T2; can use lock escalation to decide which specific tuples to get locks on (rather than locking the entire R).

113

Deadlock Prevention

84

- When there is a high level of lock contention and an increased likelihood of deadlocks
- Prevent deadlocks by giving each Xact a priority
- Assign priorities based on timestamps.
  - Lower timestamp indicates higher priority
  - i.e., oldest transaction has the highest priority
  - Higher priority Xacts cannot wait for lower priority Xacts (or vice versa)

84

Deadlock Prevention (cont'd)

85

- Assume Ti wants a lock that Tj holds. Two policies are possible:
  - **Wait-Die:** If Ti has higher priority, Ti waits for Tj; otherwise Ti aborts
  - **Wound-wait:** If Ti has higher priority, Tj aborts; otherwise Ti waits
- Both schemes will cause aborts even though deadlock may not have occurred.
- If a transaction re-starts, make sure it has its original timestamp – WHY?

85

Wait-Die

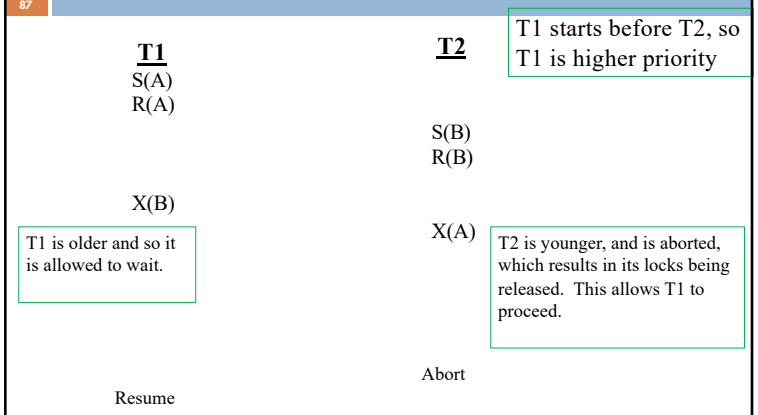
86

- Suppose Ti tries to lock an item already locked by Tj.
- If Ti is the older transaction then Ti will wait
- Otherwise Ti is aborted and re-starts later with the same timestamp.
- Lower priority transactions never wait for higher priority transactions.

86

Example: Wait-Die

87



87

Wound-Wait

88

- Suppose Ti tries to lock an item locked by Tj.
- If Ti is higher priority (older transaction)
  - then Tj is aborted and restarts later with the same timestamp;
- Otherwise, Ti waits.

88

Example: Wound-Wait

89

T1 starts before T2, so T1 is higher priority

T1	T2
S(A)	
R(A)	
	S(B)
	R(B)
X(B)	

T1 is older, so T2 is aborted, and that allows T1 to proceed. → Abort

89

Wait-Die: Let's consider…

90

T1 (ts =22) requests A: wait for T_2 or T_3 or both?

Note: ts between 20 and 25.

T2 (ts =20) wait(A) → T3 (ts =25)

90

Wait-Die: Option 1

91

One option: T_1 waits just for T_3, transaction holding lock.

But when T_2 gets lock, T_1 will have to die!

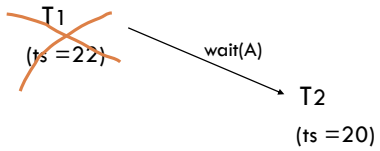
T2 (ts =20)

91

Wait-Die: Option 2

92

Another option: T_1 waits for both T_2, T_3.
T_1 allowed to wait iff there is at least one younger Xact waiting for A.
But again, when T2 gets lock, T_1 must die!

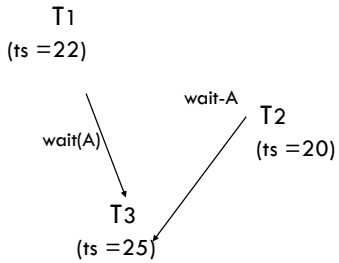


92

Wait-Die: Option 3

93

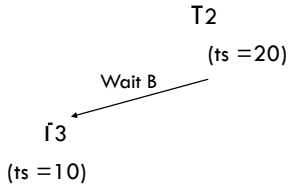
- Yet another option: T_1 preempts T_2 (T_2 is just waiting idle anyway),
- So T_1 only waits for T_3; T_2 then waits for T_1
- And lots of WFG work for Deadlock Manager



93

Wound-Wait

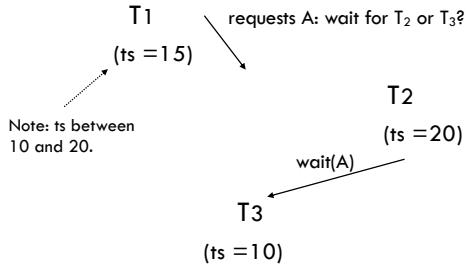
94



94

Wound-Wait: Let's Consider …

95



95

Wound-Wait: Option 1

96

One option: T_1 waits just for T_3, transaction holding lock.

But when T_2 gets lock, T_1 waits for T_2 and wounds T_2.

T1
(ts =15)

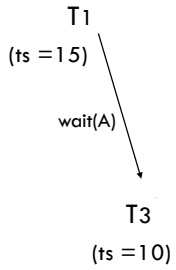
96

Wound-Wait: Option 2

97

Another option:

T_1 waits for both T_2, T_3 $\Rightarrow$ T_2 wounded right away!



97

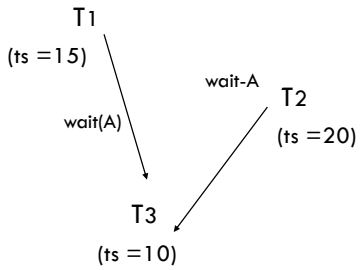
Wound-Wait: Option 3

98

Yet another option: T_1 preempts T_2, so T_1 only waits for T_3;

T_2 then waits for T_3 and T_1... $\Rightarrow$ T_2 is spared!

Lots of WFG work for Deadlock Mgr (shifting edges)



98

Comparing Deadlock Management Schemes

99

- Wait-die and Wound-wait ensure no starvation (unlike detection)
- Waits-for graph technique only aborts transactions if there really is a deadlock

99

Summary

- There are several lock-based concurrency control schemes (Strict 2PL, 2PL).
- SQL-92 provides different isolation levels that control the degree of concurrency
- The lock manager keeps track of the locks issued. Deadlocks can either be prevented or detected.

100

Q9

Question 9. Consider the relation R with schema R(A,B,C,D), and functional dependencies $\{AB \to C, C \to D, D \to A\}$.

(a) Is the relation in BCNF? Is it in 3NF? Explain why or why not.

(b) Is the decomposition of R into AB, BC and CD lossless? Why or why not?

(c) Is the decomposition of R into AB, BC and CD dependency preserving? Why or why not?

a) The keys are AB, BC, BD (by augmentation of FDs). Hence, the latter two dependencies violate BCNF but not 3NF (since all attributes are prime). R is in 3NF not BCNF.

b) No, since B is not a key for AB nor BC.

c) No, AB→C is lost.

1

Q10

Prove the following inference rule for functional dependencies using only Armstrong's axioms: If $P \to QR$ and $R \to S$, then $P \to QS$.

Show the steps of your proof, and indicate which of Armstrong's axioms is applied in each step.

$P \to QR$	Given
$R \to S$	Given
$P \to Q$	decomposition
$P \to R$	decomposition
$P \to S$	transitivity w/ $R \to S$
$P \to QS$	union

2

Q11

- For each schedule indicate which properties hold.

a) $R_1(A), W_2(A), R_1(B), Commit_1, W_3(B), R_3(B), W_3(A), Commit_3, R_2(C), Commit_2.$

Conflict serializable	Yes
ACA	Yes
Recoverable	Yes
2PL	No
Strict 2PL	No

3

Q11

- For each schedule indicate which properties hold.

b) $R_1(A), W_2(B), R_1(B), Commit_1, Commit_2$

Conflict serializable	Yes
ACA	No
Recoverable	No
2PL	Yes
Strict 2PL	No

c) $R_1(A), W_2(B), R_1(B), Commit_2, Commit_1$

Same as above except this schedule is recoverable.

4

Q12 (see practice final for full details)

- Give the sequence of lock requests using Wound-Wait policy.

(a) $R_1(X), W_2(Y), W_2(X), W_3(Y), W_1(Y)$

- T1 acquires shared lock on X; T2 acquires an exclusive lock on Y.
- T2 requests an exclusive lock on X. Since T2 is lower priority, it will wait.
- T3 requests an exclusive lock on Y; it also waits.
- T1 now requests an exclusive lock on Y; since it has the higher priority than T2, it will abort T2.

5

Q12

- Given the following actions, under strict 2PL with deadlock detection, is there a deadlock? If so, show the WFG.

$R_1(X), W_2(Y), W_2(X), W_3(Y), W_1(Y), Commit_1, Commit_2, Commit_3$

- Yes, deadlock exists.

6

FINAL REVIEW

7

Final Exam

8

- December 21, 2021 at 9:00am (2.5 hours)

8

Relational Model

9

- Logical model, physical model
- Data Independence
- Schemas
- Integrity Constraints (tuple, domain)
- Keys (superkey, PK, FK)
- Referential integrity (what is it, enforcement)

9

E-R Model

10

- Read and interpret an ER diagram
- How to translate English requirements to an ER diagram
- Avoid redundancy
- Different types of relationships (one-many, many-many, one-one)
- Total vs. partial participation
- Weak entities
- ISA hierarchies
- Covering and overlap constraints

10

SQL

11

- DDL, DML
- Create table, update, delete statements
- Relational predicates, clauses, operators
- Joins (outer, full, equijoin, self), aggregation, grouping, sub-queries, etc.
- Keys: PKs, FKs, referential integrity (ways of enforcement)
- Bag semantics vs. set semantics
- Given a schema:
  - Evaluate the results (output) of an SQL query
  - Translate English statement to write an SQL query

11

Views and Indexes

12

- View definition
- Distinction between virtual vs. materialized views
  - Benefits, disadvantages of each
- Insertions and updates on views
- Clustered vs. unclustered index
- B+ tree index, hash index, composite index
- When are indexes best used
- How to select the best index for a workload

12

Relational Algebra

13

- RA operators and operands (selection, projection, joins, renaming, set operations, and others…)
- Set vs. bag semantics
- Extended operators
- Given a schema, know how to:
  - Write an RA expression from English statement
  - Evaluate an RA expression for its output

13

Database Design

14

- Redundancy, and how this causes anomalies
- Functional Dependencies (FDs)
- Keys, superkeys
- Armstrong's Axioms
- Dependency inference
- Closure
- Minimum Cover

14

Database Design (cont'd)

15

- Projection of FDs
- Given R, and set of FDs F, find the keys
- Schema decomposition (properties, goals)

15

Normalization

16

- Lossless join decomposition
  - What does this mean
  - Test to determine if a decomposition is lossless
- Dependency preserving
  - What does this mean
  - How to check if a decomposition is dependency preserving
- BCNF, 3NF
  - Distinction between the two
  - Properties of each
  - Is a decomposition BCNF or 3NF?
  - Find a BCNF, 3NF decomposition: decomposition algorithms

16

Transactions

17

- Transaction properties (ACID)
- Schedules
  - properties such as: serial, equivalent, serializable, conflict serializable, avoid cascading aborts, recoverable, 2PL, strict 2PL
  - How to check for these properties
- Conflict operations
- Precedence graph
- Given a schedule, determine its properties

17

Locking

18

- Types of locks
- Strict 2PL, 2PL
- Phantom problem
- Performance/overhead of locks
- Isolation levels

18

Locking

19

- Deadlocks
  - Detection: Waits-for-graph
  - Prevention: Wait-die, Wound-wait
- Multiple Granularity Locking
  - Intention locks, lock conversions (upgrades/downgrades)

19

20

20

21

21

Course Evaluations

22

Final reminder if you can please complete by end of today – thank you!

evals.mcmaster.ca

22