

# 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](mailto:alipoum@mcmaster.ca))
  - Levin Noronha ([noroni@mcmaster.ca](mailto:noroni@mcmaster.ca))
  - Lucia Cristiano ([cristial@mcmaster.ca](mailto: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](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](mailto: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

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

We will be using IBM DB2

"Accessing  
DB2 Servers"  
in Avenue

9

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

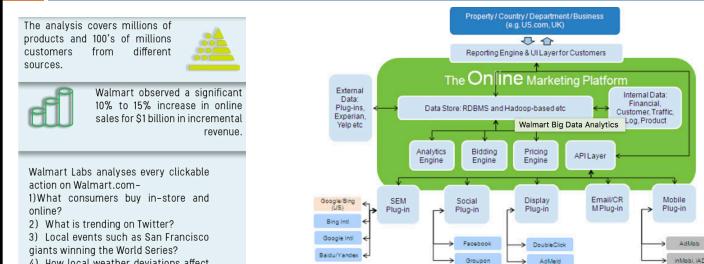
**Walmart**

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...



13

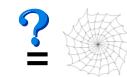
## What Is a Database System?

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 Cross](#)  
<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.miaminewstimes.com/news/miami-groups-launch-fundraiser...> ▾  
 6 hours ago · Donations are also being accepted online at the Hurricane Dorian Relief Fund organized by The Smathers Trust, a Miami-based nonprofit that ...

[Here's how to donate, help Bahamians recover from Dorian - WPLG](#)  
<https://www.local10.com/weather/hurricane/heres-how-to-donate-h...> ▾  
 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 - YouTube](#)  
<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://bloodconnection.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.newspress.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 |...](#)  
<https://kron4news.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:  
 $dapp/dt \ll dplatform/dt$

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:  
 $dapp/dt \ll dplatform/dt$

2

## Concurrency Control

3

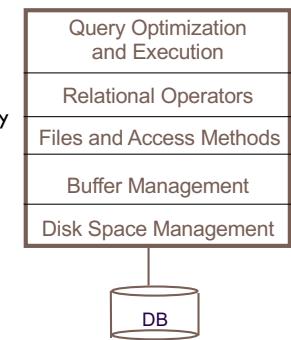
- Concurrent execution of user programs: key to good DBMS performance.
  - Disk accesses frequent
  - Keep the CPU working on several programs concurrently.
- Interleaving actions of different programs: trouble!
  - e.g., account-transfer & print statement at same time
- DBMS ensures such problems don't 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

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  $dapp/dt \ll dplatform/dt$
- 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

8

- 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

9

- 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

10

- 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

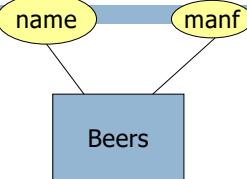
12

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

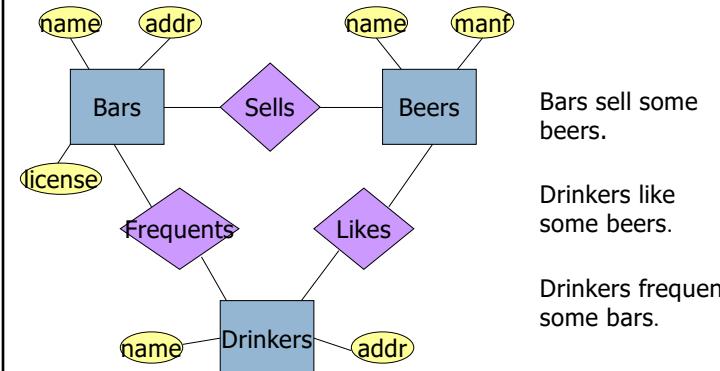
14

- 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



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

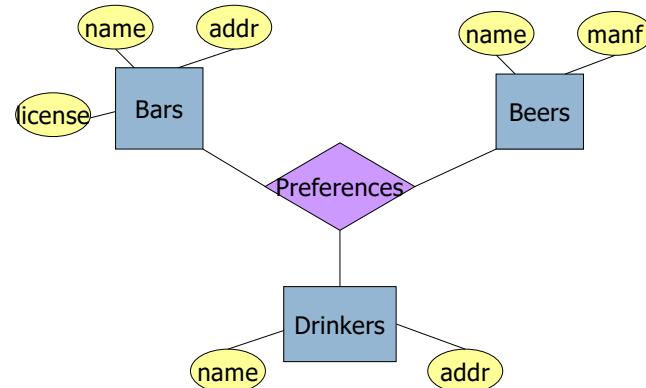
## Multiway Relationships

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

### A Typical Relationship Set

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

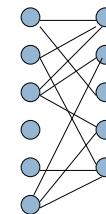
20

### Many-Many Relationships

- 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:



many-many

Note: each line is an instance of the binary relationship

22

## Many-One Relationships

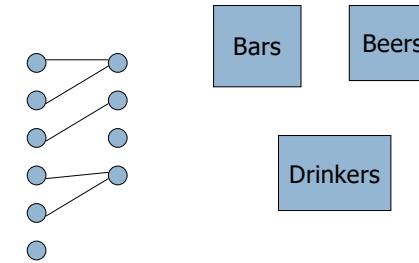
23

- 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

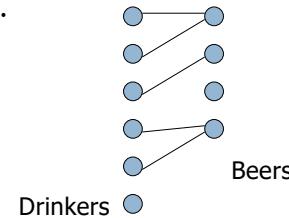
(Partial) Function on entity set

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.

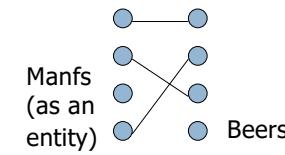


25

## 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

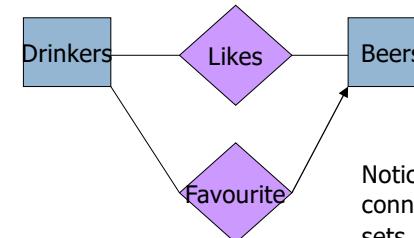
- Show a many-one relationship by an arrow entering the “one” side.
  - “at most one”
- Show a one-one relationship by arrows entering both entity sets.

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

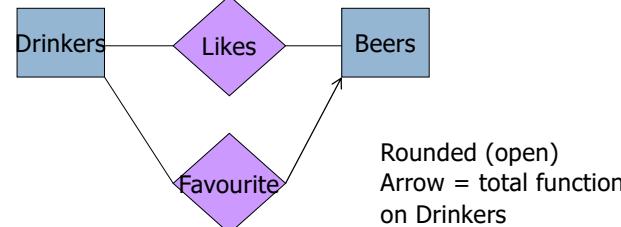


Notice: two relationships connect the same entity sets, but are different.

28

## Example: Many-One Relationship

29



29

## Example: One-One Relationship

30

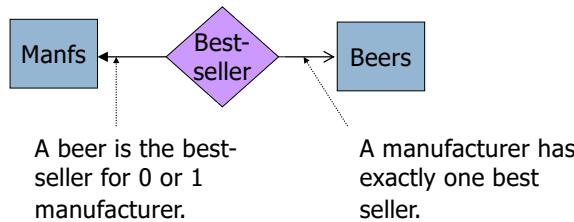
- Consider Best-seller between Manfs and Beers.
- Some beers are not the best-seller of any manufacturer
- But a beer manufacturer has to have a best-seller.



30

## In the E/R Diagram

31



31

## Participation Constraints

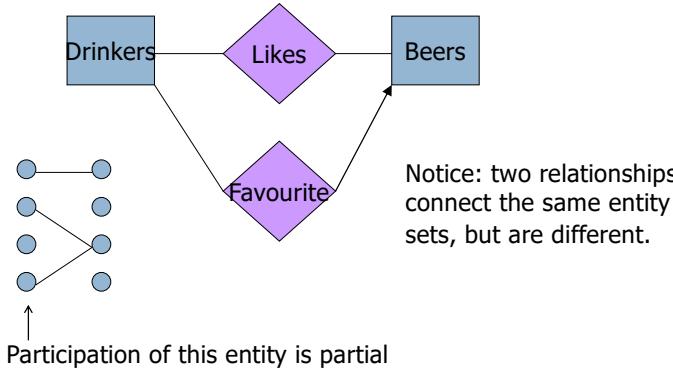
32

- 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

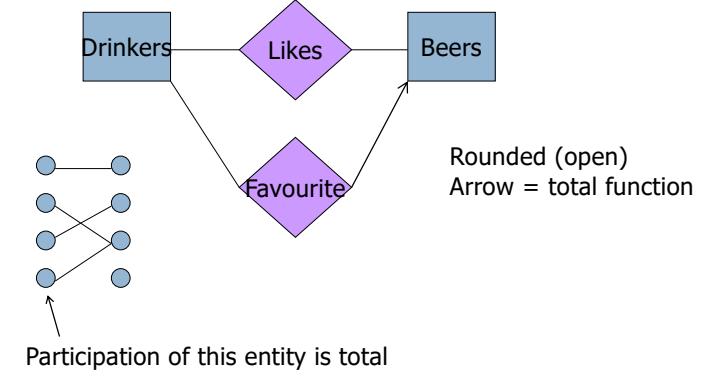
33



33

## Example: Many-One Relationship

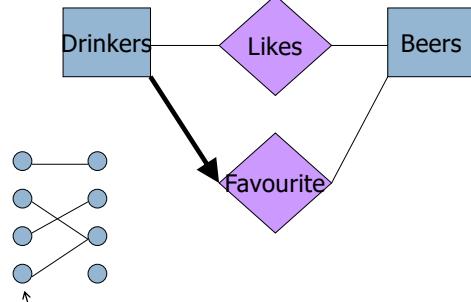
34



34

## Alternative (Textbook) Notation

35



Participation of this entity is total

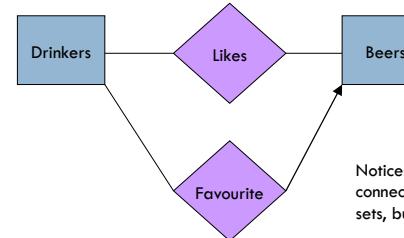
35

## 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

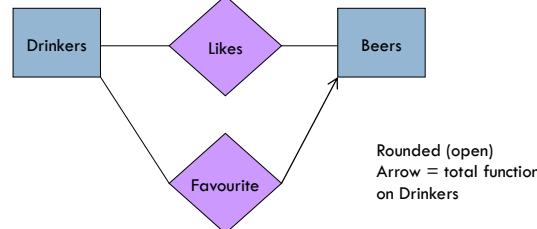
## Example: Many-One Relationship



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

2

## Example: Many-One Relationship



3

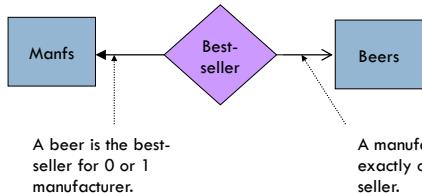
## Example: One-One Relationship

- 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



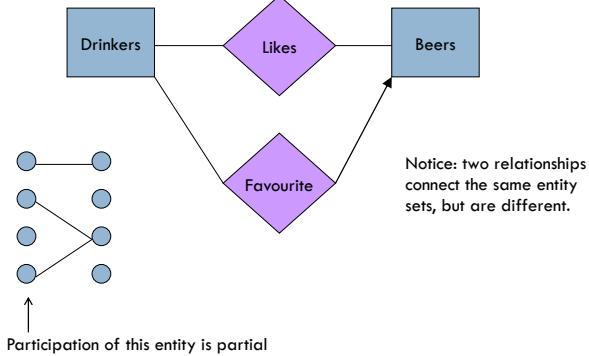
5

## 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

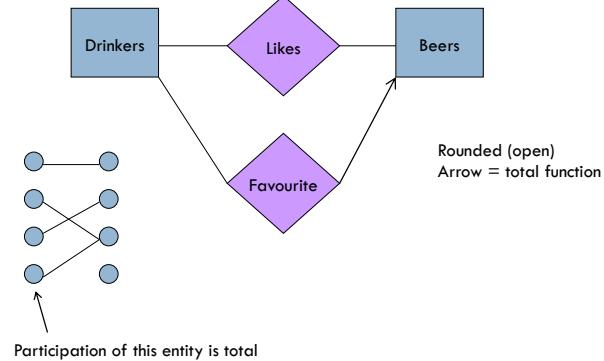
6

## Example: Many-One Relationship



7

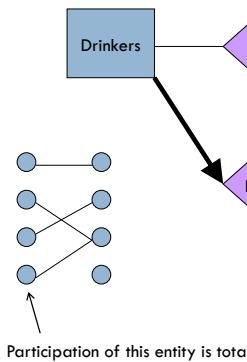
## Example: Many-One Relationship



8

## Alternative (Textbook) Notation

9

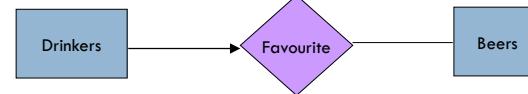


## Notation

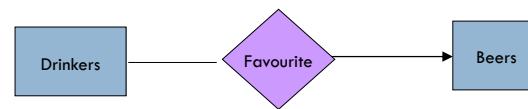
10

- Be consistent with your chosen notation!

textbook



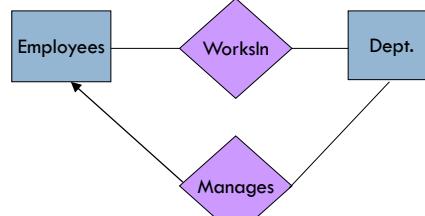
slides



10

## Key Constraints

11

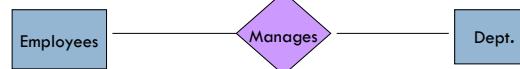


- Many-many: "An employee can work in many depts, and a dept. can have many employees"
- One-many: A dept has **at most one** manager, and employees can manage many departments

## 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**)



— Total participation (all)

→ Key constraint (at most one)

[textbook] → } Total participation and key constraint  
[slides] → } (all and exactly one)

11

12

## Attributes on Relationships

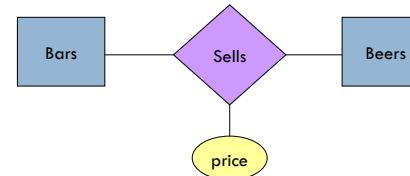
13

- 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

Relation Name	
Attribute Names	
Tuples (Records)	
ID	Name
2225555	Peter Jones
1234567	Amber Smith

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

<i>ID</i>	<i>name</i>	<i>dept_name</i>	<i>salary</i>
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

©Silberschatz, Korth and Sudarshan

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

©Silberschatz, Korth and Sudarshan

22

## Database Schemas in SQL

23

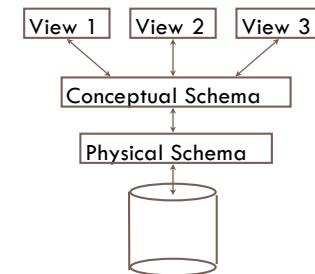
- SQL is primarily a query language, for getting information from a database.
- But SQL also includes a **data-definition** component for describing database schemas.

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/quered 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

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

**Enrolled(sid, cid, grade)**

**Enrolled(sid, cid, grade)**

**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

<i>sid</i>	<i>cid</i>	<i>grade</i>
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

<i>sid</i>	<i>name</i>	<i>login</i>	<i>age</i>	<i>gpa</i>
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

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

## 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

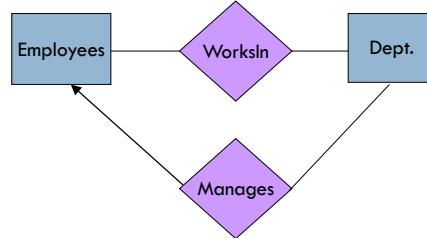
  

Cars	Registration	Dept	Owner	...
	7122 HT	75	Cordon Edouard	...
	5694 FR	93	Latour Hortense	...
	6544 XY	47	Mimault Bernard	...

33

## Key Constraints

37



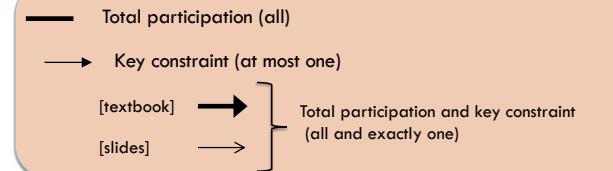
- Many-many: "An employee can work in many depts, and a dept. can have many employees"
- One-many: A dept has **at most one manager**, and employees can manage many departments

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**)



38

## Attributes on Relationships

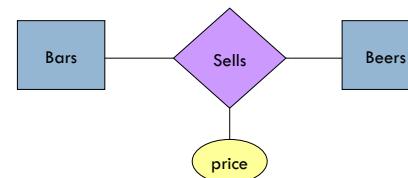
39

- 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

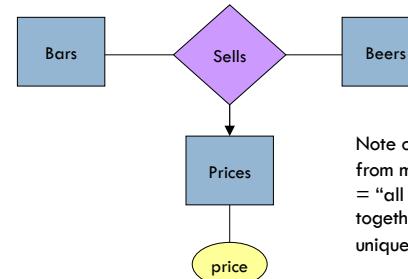
41

- 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



Note convention: arrow from multiway relationship = "all other entity sets together determine a unique one of these."

42

## Roles

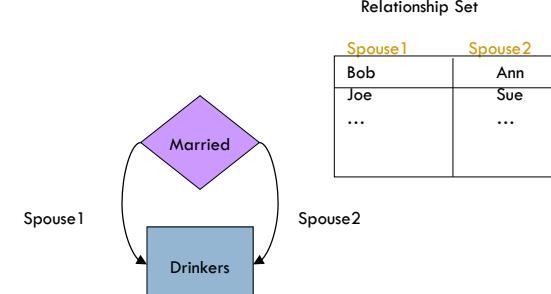
43

- 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

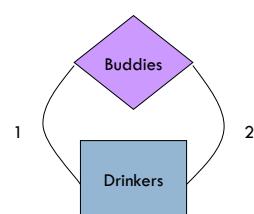
44



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

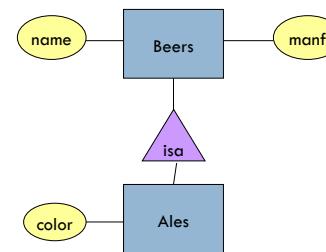
47

- **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

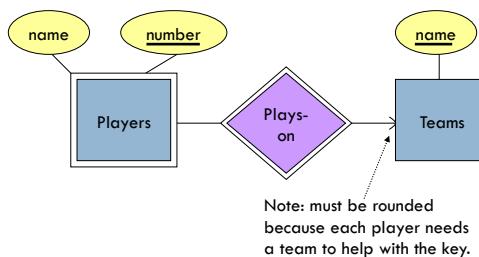
54

- **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

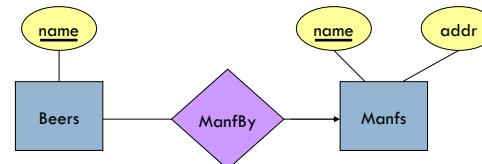
59

- **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

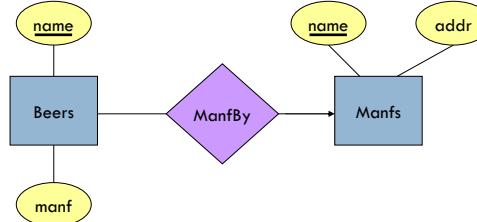


This design gives the address of each manufacturer exactly once.

60

### Example: Bad

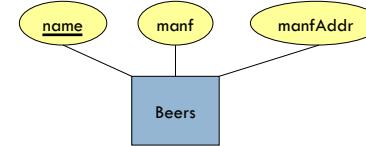
61



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

### 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.

61

62

## 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

- There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes.
- 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  $R = \text{Sells}$ ,  $S = \text{Beers}$ .
  - An insert or update to  $\text{Sells}$  that introduces a nonexistent beer must be rejected.
  - A deletion or update to  $\text{Beers}$  that removes a beer value found in some tuples of  $\text{Sells}$  can be handled in three ways...

11

## Actions Taken --- (2)

- 12
1. **Default** : Reject the modification.
  2. **Cascade** : Make the same changes in  $\text{Sells}$ .
    - **Deleted beer**: delete  $\text{Sells}$  tuple.
    - **Updated beer**: change value in  $\text{Sells}$ .
  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

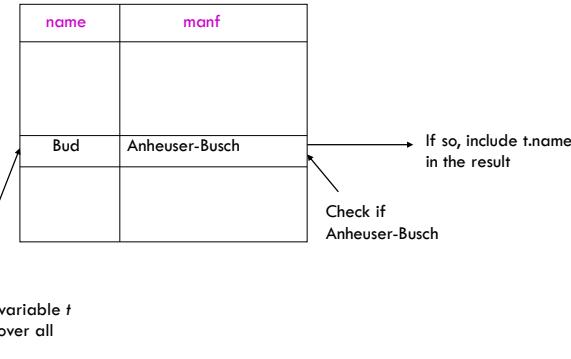
30

- 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

## Result of Query:

36

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

36

## 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

## 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 Frequent(drinker, bar), find the beers liked by at least one person who frequents Joe's Bar.

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

29

## Example: Joining Two Relations

30

- Alternatively can use explicit (named) tuple variables

```
SELECT beer
FROM Likes l, Frequent 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

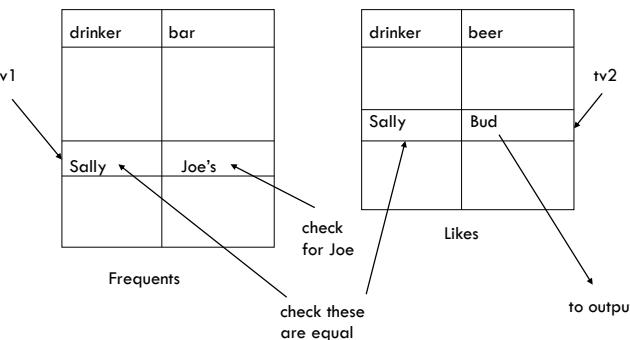
32

- 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.

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

Drinkers who frequent Joe's Bar

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

```
SELECT bar
FROM Sells
```

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

```
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
```

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

Sells(bar, beer, price)

```
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

## 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,  $\geq \$90,000$  and  $\leq \$100,000$ )
 

```
select name
        from instructor
       where salary between 90000 and 100000
```

43

## The Operator ANY

44

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

44

## The Operator ALL

45

- $x <> \text{ALL}(<\text{subquery}>)$  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:  $x \geq \text{ALL}(<\text{subquery}>)$  means there is no tuple larger than  $x$  in the subquery result.

45

## Example: ALL

46

- From  $\text{Sells}(\text{bar}, \text{beer}, \text{price})$ , 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

- $<\text{value}> \text{IN}(<\text{subquery}>)$  is true if and only if the  $<\text{value}>$  is a member of the relation produced by the subquery.
  - Opposite:  $<\text{value}> \text{NOT IN}(<\text{subquery}>)$ .
- 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);
```

One loop, over the tuples of R

Two 2's

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

| a | b |
|---|---|
| 1 | 2 |
| 3 | 4 |

| b | c |
|---|---|
| 2 | 5 |
| 2 | 6 |

R S

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 |

| b | c |
|---|---|
| 2 | 5 |
| 2 | 6 |

R 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 (

```

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

```

    SELECT *
    FROM Beers
    WHERE manf = b1.manf AND
          name <> b1.name);

```

Notice the SQL "not equals" operator

3

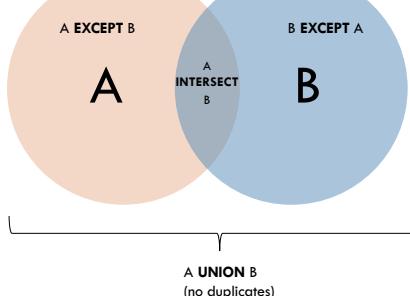
## Union, Intersection, and Difference

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

## Example: Intersection

- 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

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

subquery is really a stored table.

The drinker frequents a bar that sells the beer.

7

## Bag Semantics

- 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 &amp; 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 forgot 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
```

$$\vdots$$

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.

## Deletion

27

- To delete tuples satisfying a condition from some relation:

```
DELETE FROM <relation>
WHERE <condition>;
```

26

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

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

Sells(bar, beer, price)

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:
  - Aggregated, or
  - 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
```

Result

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

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  |

{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

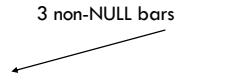
## Solution

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

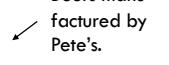
```
SELECT beer, AVG(price)
FROM Sells
GROUP BY beer
HAVING COUNT(bar) >= 3 OR
```

```
beer IN (SELECT name
          FROM Beers
          WHERE manf = 'Pete''s');
```

Beer groups with at least  
3 non-NULL bars



Beers man-  
ufactured 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