PSTAT10_lecturenote

Hongjie Lin

Lecture 14 / November 12

DATABASE MANAGEMENT SYSTEM (DBMS): software for managing databases and providing access to them.

- responds to instructions given by application programs, executing on behalf of users.
- written in the database language of the DBMS; for us it's SQL, where responses include results of queries

DATABASE: an organized, machine-readable collection of symbols, to be interpreted as a true account of some enterprise.

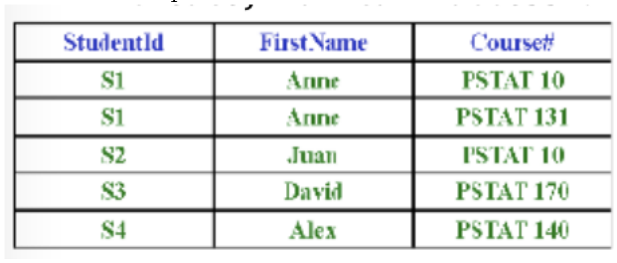
- machine-updatable, so a database is a collection of **VARIABLES** | typically avail to a group of users
- a collection of data that is logically coherent

RELATIONAL DATABASE stores and manages structure data

- a database of structured data (data that can be structured in tables) organized into a collection of relations

STRUCTURED DATA - e.g. band statement, address books is stored in relational database

- to manage all this structure data a relational database management system is used to create, maintain access & manipulate data



StudentId	FirstName	Course#
S1	Anne	PSTAT 10
S1	Anne	PSTAT 131
S2	Juan	PSTAT 10
S3	David	PSTAT 170
S4	Alex	PSTAT 140

- to be interpreted as a true account
  - e.g. interpretation: "Student S1, whose first name"Anne" is enrolled in course "Pstat 10"

RELATIONAL MODEL: a theoretical model of relational databases; based on set theory, it is mathematically sound

- still the leading model | most widely used data model for commercial data-processing

- simple and easy to maintain
- a **DATA MODEL**
  - used to rep data and the relationships between data items
- specifies the ways in which the data can be used

3 PARTS of the RELATIONAL MODEL

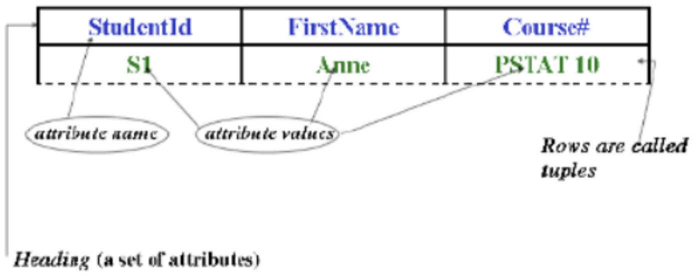
1. **STRUCTURAL:** the building blocks from which databases are constructed.

- Relation, attribute, domain, tuple, keys (super key, primary key, candidate key, foreign key)

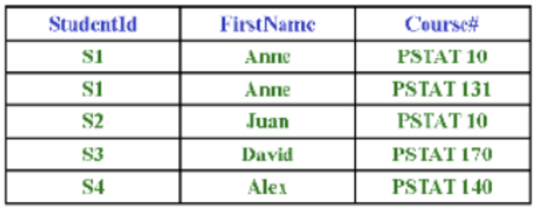
2. **INTEGRITY:** a collection of rules that all databases must obey.
3. **MANIPULATIVE:** includes operations for retrieving data, for updating the database.

RELATION - order doesn't matter (e.g. S1 and PSTAT 131 can in any order)

- NAME - each relation in a relational database must use a unique name
  - e.g. ENROLLMENT
- **TUPLES** - rows, the number of records.
- defined on a number of **ATTRIBUTES** (e.g. the Heading is a set of attributes, there are attribute values for each tuple)



Updating



StudentId	FirstName	Course#
S1	Anne	PSTAT 10
S1	Anne	PSTAT 131
S2	Juan	PSTAT 10
S3	David	PSTAT 170
S4	Alex	PSTAT 140

ENROLLMENT is a variable. Another row has been added, thus the variable has updated

Domains
each attribute of a relation is associated with a domain

- e.g. Course_ID could be the domain associated with Course#
- TUPLES: the rows of a relationship, other than the header row containing the attribute names

- a relation is a set of tuples

KEYS
an important part of the structural component of the relational model

- e.g. Course_ID could be the domain associated with Course#
- role - if you know the value of attribute A, you can determine the value of attribute B
- primary, super key, candidate, foreign

PRIMARY KEY
an attribute (or a set of attributes) that uniquely IDs any tuple in a given relation

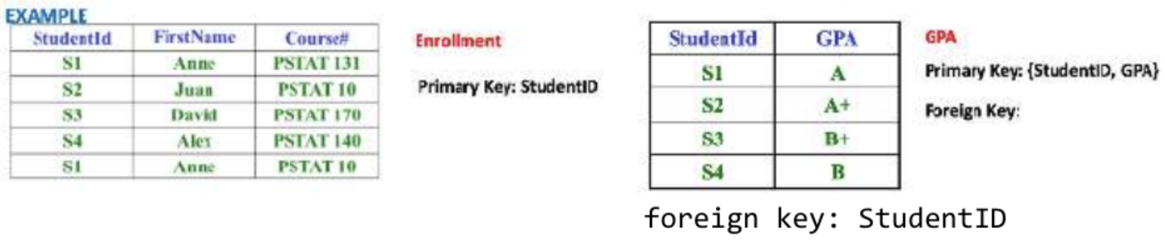
- for any relation we will choose ONE primary key

- chosen to have as few attributes as possible

- the same primary key cannot be used for different relations
- values cannot be null
- e.g. StudentID is the primary key for ENROLLMENT

CANDIDATE KEY
used for choosing a primary key; there may be more than 1 choice of primary key for a relation

- candidate keys are selected from the set of super keys. Candidate keys should not have any redundant at-



EXAMPLE

StudentId	FirstName	Course#
S1	Anne	PSTAT 131
S2	Juan	PSTAT 10
S3	David	PSTAT 170
S4	Alex	PSTAT 140
S1	Anne	PSTAT 10

Enrollment

Primary Key: StudentID

StudentId	GPA
S1	A
S2	A+
S3	B+
S4	B

GPA

Primary Key: {StudentID, GPA}

Foreign Key:

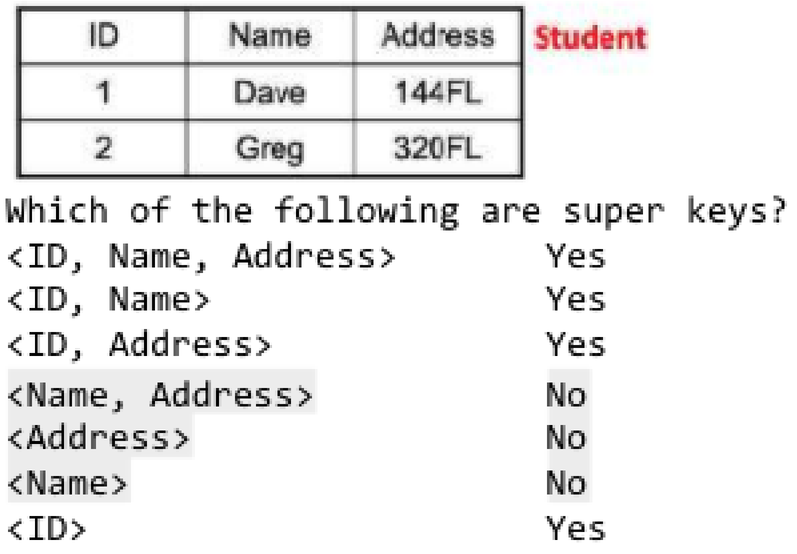
foreign key: StudentID

  tributes.

Which student enrolled in PSTAT 10 has a GPA of A+ or A?

- you need both relations to cross reference them

KEYS - another example



ID	Name	Address
1	Dave	144FL
2	Greg	320FL

Student

Which of the following are super keys?

<ID, Name, Address>	Yes
<ID, Name>	Yes
<ID, Address>	Yes
<Name, Address>	No
<Address>	No
<Name>	No
<ID>	Yes

KEY ROLE: if u know the value of attribute A, u can determine the value of attribute B

INTEGRITY of the RELATIONAL MODEL

- a set of rules that all relational databases must obey
- required to ensure that the data base is consistent and complete
- 2 general integrity rule:
  - entity integrity rule
  - referential integrity rule

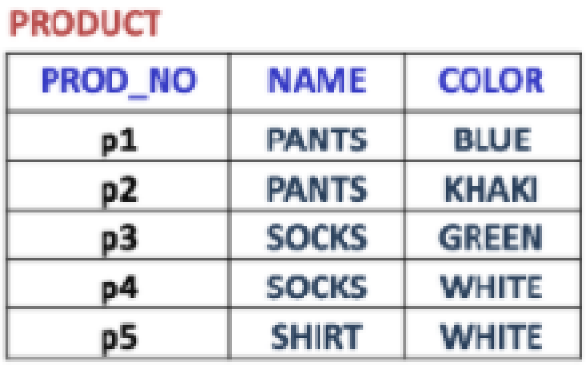
ENTITY INTEGRITY RULE

- specifies that the primary key for each relation must be unique and that the primary key must have values other than NULL for each attribute (a null-value represents that no value has been assigned to an attribute)

- ensures that there are no *duplicate* records within a given relation

justification: we must be able to identify each tuple uniquely

e.g.



PRODUCT

PROD_NO	NAME	COLOR
p1	PANTS	BLUE
p2	PANTS	KHAKI
p3	SOCKS	GREEN
p4	SOCKS	WHITE
p5	SHIRT	WHITE

Assume: Primary key: PROD_NO

Suppose we have 2 new products to add to the above relation. Both are blue socks, but one is wool, the other cotton. They have yet to be assigned product numbers.

Breach of entity integrity rule

- for wool blue socks add the tuple
  <null, SOCKS, BLUE>

- for cotton blue socks add:
  <null, SOCKS, BLUE>

These 2 tuples are not uniquely identifiable, so integrity rule has been breached

- if we assign product number p6 to wool blue socks and p7 to cotton blue socks, integrity is maintained

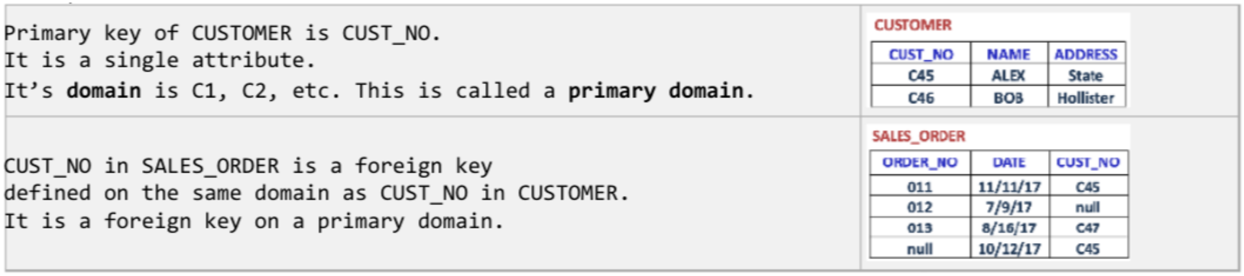
SUMMARY: ENTITY INTEGRITY:
The entity Integrity Rule states that for every instance of an entity, the value of the primary key must exist, be unique, and cannot be null. Without entity integity, the primary key could not fulfill its role of uniquely indentifying each instance of an entity.

REFERENTIAL INTEGRITY

- constraint involving two relations

- requires all foreign key

Example
ensures that only customers whose details are in the database can place orders. (name, address etc...)



Primary key of CUSTOMER is CUST_NO.
It is a single attribute.
It's **domain** is C1, C2, etc. This is called a **primary domain.**

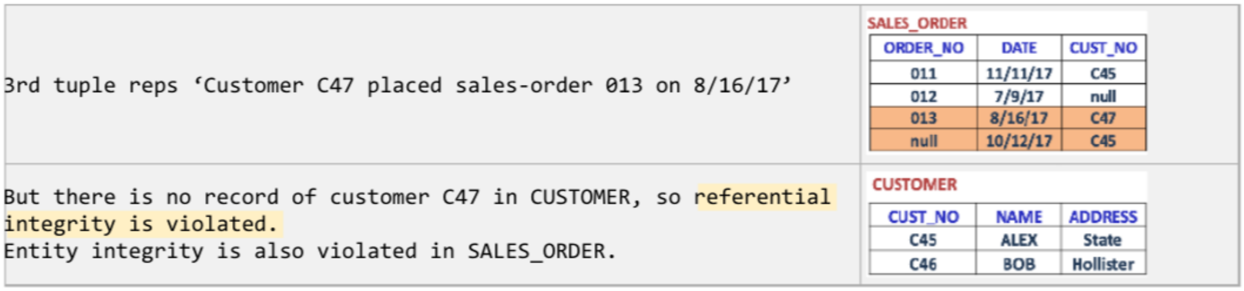
CUSTOMER

CUST_NO	NAME	ADDRESS
C45	ALEX	State
C46	BOB	Hollister

CUST_NO in SALES_ORDER is a foreign key
defined on the same domain as CUST_NO in CUSTOMER.
It is a foreign key on a primary domain.

SALES_ORDER

ORDER_NO	DATE	CUST_NO
011	11/11/17	C45
012	7/9/17	null
013	8/16/17	C47
null	10/12/17	C45



3rd tuple reps 'Customer C47 placed sales-order 013 on 8/16/17'

SALES_ORDER

ORDER_NO	DATE	CUST_NO
011	11/11/17	C45
012	7/9/17	null
013	8/16/17	C47
null	10/12/17	C45

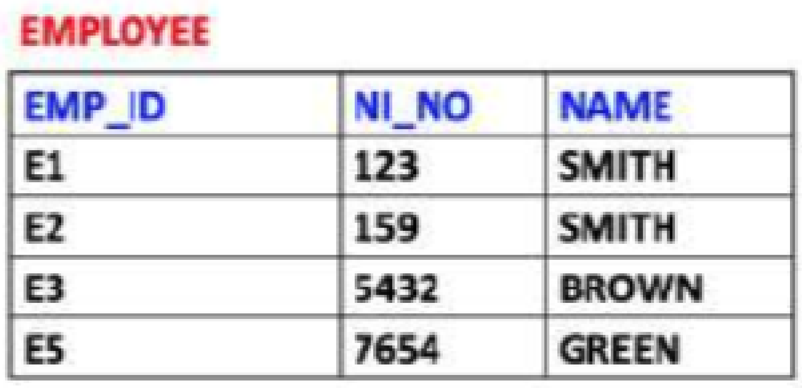
But there is no record of customer C47 in CUSTOMER, so referential integrity is violated.
Entity integrity is also violated in SALES_ORDER.

CUSTOMER

CUST_NO	NAME	ADDRESS
C45	ALEX	State
C46	BOB	Hollister

SUMMARY: REFERENTIAL INTEGRITY
The referential integrity rule states that every foreign key attribute must match a primary key attribute in an associated relation. Referential integrity ensures that we can correctly navigate between related entities.

Lecture 15 / Nov 14



EMPLOYEE

EMP_ID	NI_NO	NAME
E1	123	SMITH
E2	159	SMITH
E3	5432	BROWN
E5	7654	GREEN

```
**SUPER KEYS**            **CANDIDATE KEYS**
{EMP_ID,NI_NO,NAME}         {EMP_ID,NI_NO}
{EMP_ID,NAME}               {EMP_ID}
{EMP_ID,NAME}               {NI_NO}
{NI_NO,NAME}
{EMP_ID}
{NI_NO}
```

RELATION SCHEMA
schema of a relation consists the relation name and the name of its attributes
e.g. Product(PName, Price, Category, Manufacturer)

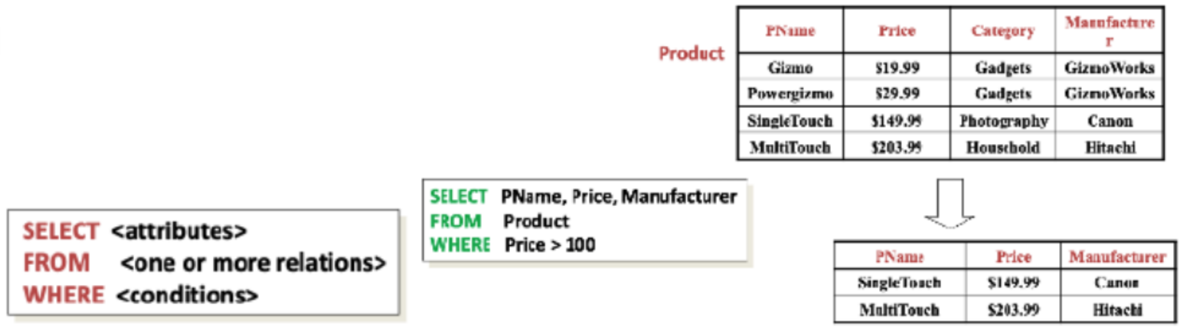
- Primary key is PName

MANIPULATION of relational standard language for storing, querying, and manipulating data: Structured Query Language (SQL)

SQL

Data Manipulation Language (DML) Data Definition Language (DDL)

SQL Query



SELECT <attributes>
FROM <one or more relations>
WHERE <conditions>

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Houshold	Hitachi

```
SELECT PName, Price, Manufacturer
FROM   Product
WHERE  Price > 100
```

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

SQLite

- The most widely used database engine in the world.
- built into most mobile phone and most computers and comes bundled inside countless other applications that people use everyday
- we'll use an SQLite database in RStudio

Course Database, Descriptions

- we'll be working with a small online clothing store called 'Tiny Clothes'

WORKSHEET 13.R

//Exercise1//

```
# TEXTBOOK SECTION 10.1

# if(condition){\ do any code here \n}   | \ means new line
myvec   <-   c(2.73,5.40,2.15,5.29,1.36,2.16,1.41,6.97,7.99,9.52)
mymat   <-   matrix(c(2,0,1,2,3,0,3,0,1,1),5,2)
if(any((myvec-1)>9)||matrix(myvec,2,5)[2,1]<=6){
  cat("Condition satisfied --\n")
  new.myvec <- myvec
  new.myvec[seq(1,9,2)] <- NA
  mylist <- list(aa=new.myvec,bb=mymat+0.5)
  cat("-- a list with", length(mylist),"members now exists.")
}
```

```
## Condition satisfied --
## -- a list with 2 members now exists.
```

//Exercise2//

```
# if(condition){ \ do any code here if the condition is FALSE\}
a <- 3
mynumber <- 4
if(a<=mynumber){
  cat("Condition was", a<=mynumber)
  a <- a^2
} else {
  cat("Condition was", a<=mynumber)
  a <- a-3.5
}
```

```
## Condition was TRUE
```

```
# TRUE the 1st run, but false the 2nd run
# if ... else can run through an entire logical vector
```

//**Exercise3**//

```
x <- 5
y <- -5:5
result <- ifelse(test=y==0,yes=NA,no=x/y)
result
```

```
##  [1] -1.000000 -1.250000 -1.666667 -2.500000 -5.000000        NA  5.000000
##  [8]  2.500000  1.666667  1.250000  1.000000
```

```
# test takes a logical-valued data structure | yes provides the lement to return if TRUE | no gives the
# the returned structure will be the same length and attributes as test
```

//**Exercise4**//

```
# Exercise 10.1 a
vec1 <- c(2,1,1,3,2,1,0)
vec2 <- c(3,8,2,2,0,0,0)
if(vec1[1]>=2&&vec2[1]>=2){
  cat("Print me!")
}
```

```
## Print me!
```

```
if(!is.na(vec2[3]))
  {cat("Print me!")}
```

```
## Print me!
```

WORKSHEET 13

the R command next halts the processing of the current iteration and advances the looping index
a) write a for loops to generate a sequence of integers from 1 to 10.

```
for(i in 1:10){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

b) use next to skip the first 5 iterations

```r
for(i in 1:10){
if(i<6){
  next
}
print(i)
}
```

```
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

//**Exercise1**//

```r
b <- 0
for(a in 1:3){
a <- a+1
cat("a equals", a, "\n")
b <- b+1
cat("b equals",b, "\n")
}
```

```
## a equals 2
## b equals 1
## a equals 3
## b equals 2
## a equals 4
## b equals 3
```

Create two numeric vectors, a and b
a) write a nested for loop where outer for loops increments a by 1 and run 3 times
and the inner for loop increments b by 1 and run three times.

```r
b <- 0
a <- c(1,2,3)
b <- c(4,5,6)

for(a in 1:3){
  a <- a + 1
  for(j in 1:3){
    b = b + 1
    if (b==2){
      break
    }
  }
}
```

```
## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used

## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used
## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used

## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used

## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used

## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used

## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used

## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used

## Warning in if (b == 2) {: the condition has length > 1 and only the first
## element will be used
```

//**Exercise2**//
Write a function that returns the cube of a positive number

```r
Cube_it <- function(t){
  if(t>0){
    r <- t^3
    return(r)
  } else{
    cat('Your number must be positive')
  }

}
Cube_it(-2)
```

```
## Your number must be positive
```

//**Exercise3**//
Write a function to sum the first n natural numbers

```r
Sum_numbers <- function(n){
  if(n<10){
  s <- (n*(n+1))/2
  return(s)
  }else{
  cat("your number must be less than 10")
  }


}
Sum_numbers(3)
```

```
## [1] 6
```

```r
Sum_numbers(11)
```

```
## your number must be less than 10
```

//Exercise4//
Write a repeat loop to print 'PSTAT 10' five times

```r
v <- 0
repeat {
  v <- v + 1
  print("PSTAT 10")
  if(v==5){
    break
  }
}
```

```
## [1] "PSTAT 10"
## [1] "PSTAT 10"
## [1] "PSTAT 10"
## [1] "PSTAT 10"
## [1] "PSTAT 10"
```

```r
for(v in 1:5){
  cat("PSTAT 10")
}
```

```
## PSTAT 10PSTAT 10PSTAT 10PSTAT 10PSTAT 10
```

//Exerciise5//
a) How many even numbers in a given vector? x <- c(2,5,3,9,8,11,6)

```r
x <- c(2,5,3,9,8,11,6)

count <- 0
for(i in x){
  if(i %% 2== 0){
    count = count + 1
  }
}
print(count)
```

```
## [1] 3
```

b) How many even and odd together in a given vector? x <- c(2,5,3,9,8,11,6)

```r
countodd <- 0
counteven <- 0
for(i in x){
  if(i %% 2 != 0) countodd = countodd+1
  if(i %% 2 == 0) counteven = counteven+1
}
print(countodd) #4
```

```
## [1] 4
```

```r
print(counteven) #3
```

```
## [1] 3
```

```r
countodd <- 0
counteven <- 0
for (i in x){
  if(i %% 2 != 0)
```

```r
    countodd = countodd+1
  if(i %% 2 == 0)
    counteven = counteven+1
}

cat("number of odd numbers is",countodd, "\n the number of even numbers is", c(counteven))
```

```
## number of odd numbers is 4
##  the number of even numbers is 3
```

Lecture 15 / RSQL TUTORIAL