BZAN 6354

Lecture 14

April 22, 2024

Dr. Mark Grimes, Ph.D.
gmgrimes@bauer.uh.edu

UNIVERSITY of HOUSTON
C. T. BAUER COLLEGE of BUSINESS
Department of Decision & Information Sciences

Agenda

- This is our last regular class meeting ☹

- Administration

- Quick Review of Functional Dependencies, Armstrong’s Axioms, and Normal Forms

- Module 13.1 – String Manipulation Functions
- Module 13.2 – Dates and Times
- Module 13.3 – Hierarchical Queries
- Module 13.4 – Extended GROUP BY clauses

Break

- Wrap up and exam question opportunity

Administration

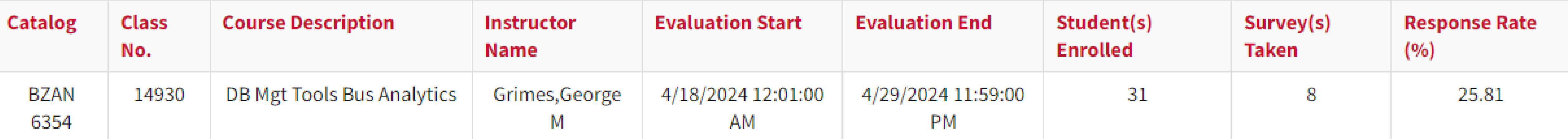
- Assignment 4 due today

- SQL project due by midnight on Friday, April 26
  - Good practice for the SQL you will write on the exam
  - Being due on Friday lets you have time to STUDY for the exam!

- Exam 2 is one week from today – Monday April 29
  - In class on paper
  - 75 minutes
  - Closed Book / Individual Effort
  - No “final exam” during exam week

Administration: Course Evaluations

- Please fill out the course evaluation!
  - As of this morning about 25% have completed
  - Goal is 60% for a half point, or 80% for a full point of extra credit!



Catalog	Class No.	Course Description	Instructor Name	Evaluation Start	Evaluation End	Student(s) Enrolled	Survey(s) Taken	Response Rate (%)
BZAN 6354	14930	DB Mgt Tools Bus Analytics	Grimes,George M	4/18/2024 12:01:00 AM	4/29/2024 11:59:00 PM	31	8	25.81

- Link is in AccessUH →

COURSE EVALUATION

Faculty/Course Evaluation

Review: Functional Dependencies

- What is a functional dependency?
  - FDs specify a relationship between attributes in a relation
  - May be semantically obvious or inferred

- In the expression A → B, what do we call A and B?
  - A: Determinant
  - B: Dependent

- Where do functional dependencies come from?
  - Business Rules

- How do you know a functional dependency is undesirable?
  - The determinant is not a candidate key

Review: Armstrong’s Axioms

- Three primary Axioms
  - Reflexivity
    - If Y is a subset of X, then X → Y (a trivial dependency)
  - Augmentation
    - If X → Y, then {X,Z} → Y and {X,Z}→{Y,Z}
  - Transitivity
    - If X → Y and Y → Z, then X → Z

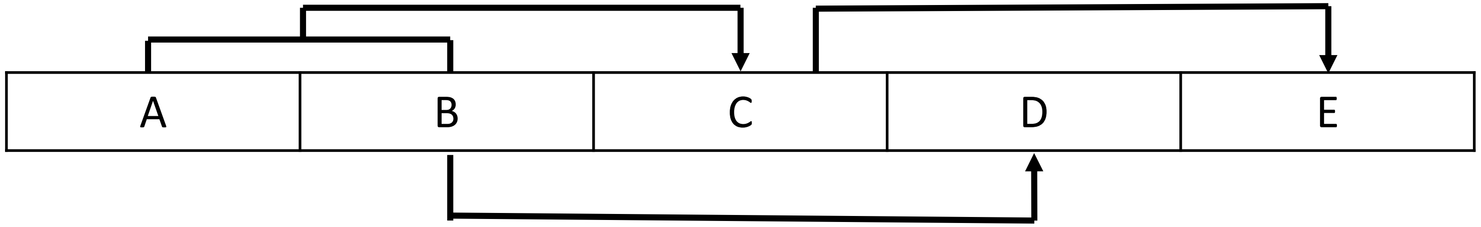
- Four inference rules
  - Union
    - If X → Y and X → Z, then X → {Y,Z}
  - Decomposition
    - If X → {Y,Z}, then X → Y and X → Z
  - Composition
    - If A → B and C → D, then {A,C} → {B,D}
  - Pseudotransitivity
    - If X → Y and {Y,W} → Z, then {X,W} → Z

Review: Normal Forms

- Normal Forms
  - 1NF
    - No multi value attributes
    - Meet the basic requirements to be a relation
  - 2NF
    - 1NF + No partial dependencies
  - 3NF
    - 2NF + No transitive dependencies
  - BCNF
    - 3NF + Every determinant must be a superkey
    - Removes the possibility of non-superkeys determining key attributes (which is OK in 3NF)

Simplest all inclusive example I can come up with

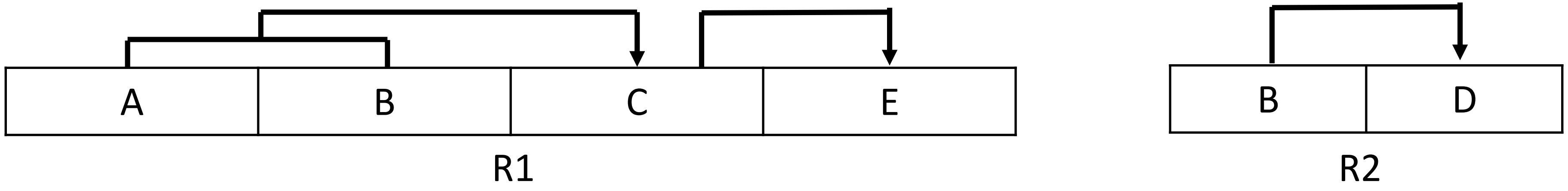
- R1(A, B, C, D, E)
  - FD1: {A,B} → C
  - FD2: B → D
  - FD3: C → E



- {A,B} is the only CK, and is thus the PK
  - You can figure this out via decomposition or synthesis
- FD2 is a partial dependency, violates 2NF
- FD3 is a transitive dependency, violates 3NF

Simplest all inclusive example I can come up with

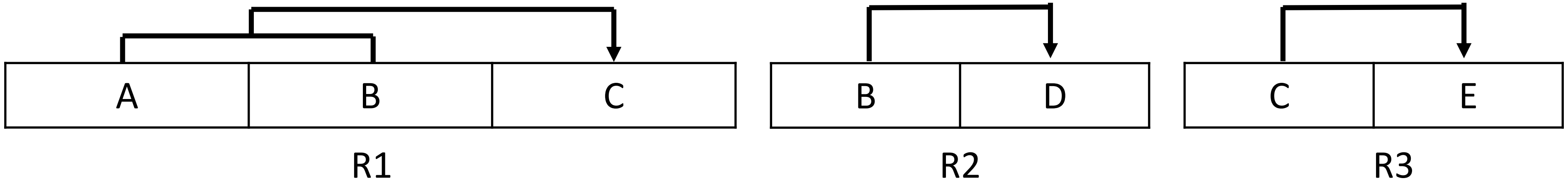
- To get to 2NF, decompose R into two relations:
  - R1(A,B,C,E)
    - FD1: {A,B} → C
    - FD2: C → E
  - R2(B, D)
    - FD3: B → D



- R1 has no partial dependencies, but still has a transitive dependency (FD2), thus is in 2NF
- R2 has no partial nor transitive dependencies, thus is in 3NF

Simplest all inclusive example I can come up with

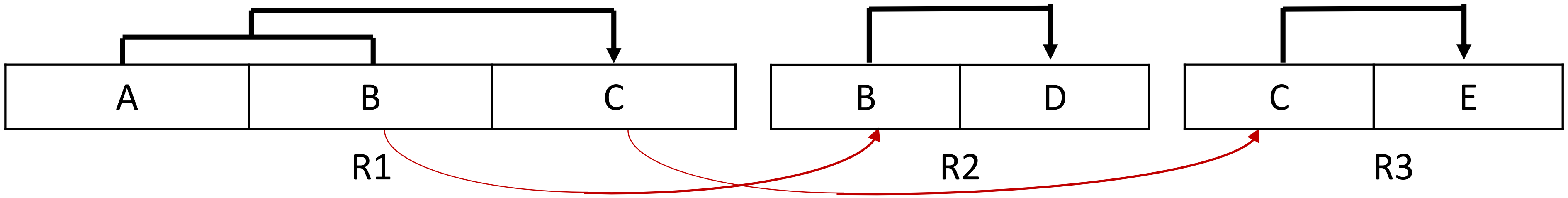
- To get to 3NF, decompose into three relations:
  - R1(A,B,C)
    - FD1: {A,B} → C
  - R2(B, D)
    - FD3: B → D
  - R3(C, E)
    - FD2: C → E



R1, R2, and R3 are all in 3NF now – Yipee!

Simplest all inclusive example I can come up with

- Note that all attributes and dependencies have been preserved, and we can recreate the original relation be joining the relations back together using the attributes that overlap in the resulting relations
  - B is now a FK in R1 and a CK in R2
  - C is now a FK in R1 and a CK in R3



On to the new!

Module 13.1

String manipulation SQL functions

- Concatenation
- SUBSTRING
- LENGTH
- TRIM (RTRIM / LTRIM)
- PAD (RPAD / LPAD)
- INSTR
- DECODE
- CASE

Before we get started: What is DUAL?

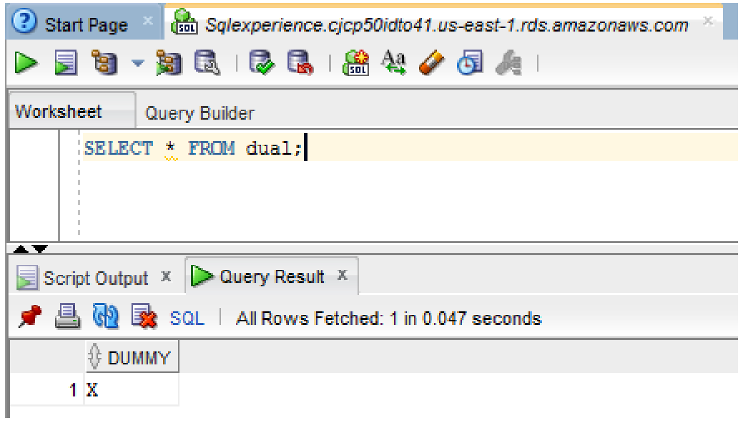
- DUAL is a table in Oracle (and some other DBMS) that you can query when you are executing queries that do not require a table, but you must specify one for parsing the query:
  - SELECT 2 + 4 From DUAL
  - Returns: 6

- Many DMBS do not require the use of dual. For example, in Microsoft SQL Server, this works fine:
  - SELECT 2 + 4
  - Returns: 6

Before we get started: What is DUAL?

- DUAL has one tuple, with one attribute named “DUMMY” and one value, “X”

- `SELECT * FROM dual`
- `SELECT dummy FROM dual`



```
SELECT * FROM dual;
```

All Rows Fetched: 1 in 0.047 seconds

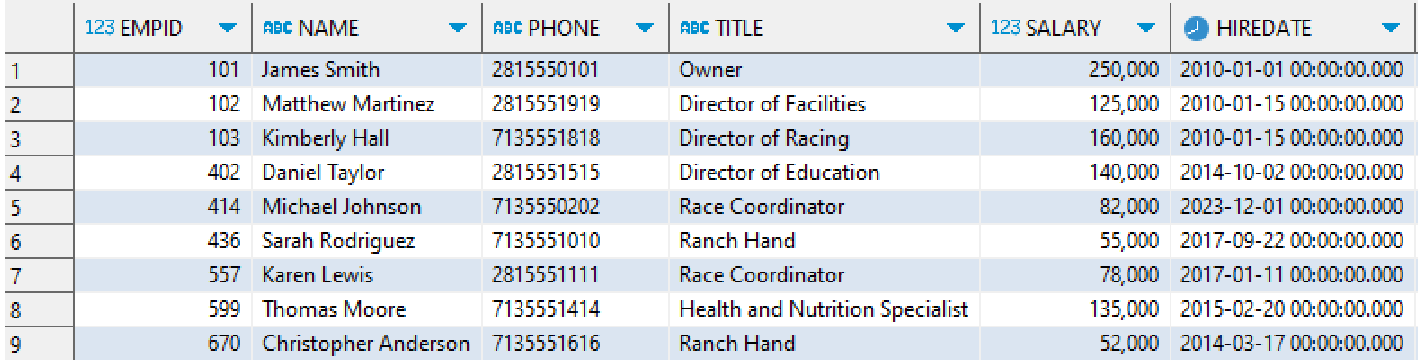
```
    DUMMY
  1 X
```

- Fun history lesson: the original implementation of dual had two tuples (hence the name “dual”) and was primarily used to double the number of records returned from a “real” table by doing the Cartesian product.

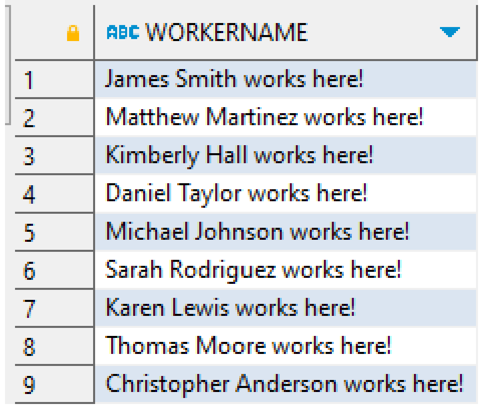
The Concatenation Operator (||)

- Allows you to join multiple strings together using two vertical bars (AKA pipes)
  - Can be values from text attributes or a literal string
- In the SQL project database, you might try (for fun):

```
SELECT name || ' works here!' AS WorkerName FROM workers;
```

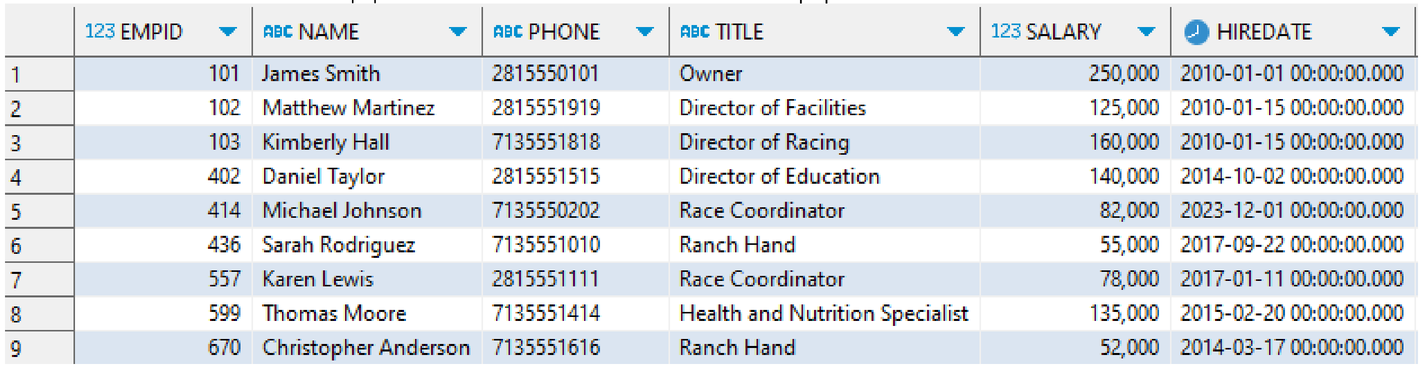


	EMPID	NAME	PHONE	TITLE	SALARY	HIREDATE
1	101	James Smith	2815550101	Owner	250,000	2010-01-01 00:00:00.000
2	102	Matthew Martinez	2815551919	Director of Facilities	125,000	2010-01-15 00:00:00.000
3	103	Kimberly Hall	7135551818	Director of Racing	160,000	2010-01-15 00:00:00.000
4	402	Daniel Taylor	2815551515	Director of Education	140,000	2014-10-02 00:00:00.000
5	414	Michael Johnson	7135550202	Race Coordinator	82,000	2023-12-01 00:00:00.000
6	436	Sarah Rodriguez	7135551010	Ranch Hand	55,000	2017-09-22 00:00:00.000
7	557	Karen Lewis	2815551111	Race Coordinator	78,000	2017-01-11 00:00:00.000
8	599	Thomas Moore	7135551414	Health and Nutrition Specialist	135,000	2015-02-20 00:00:00.000
9	670	Christopher Anderson	7135551616	Ranch Hand	52,000	2014-03-17 00:00:00.000

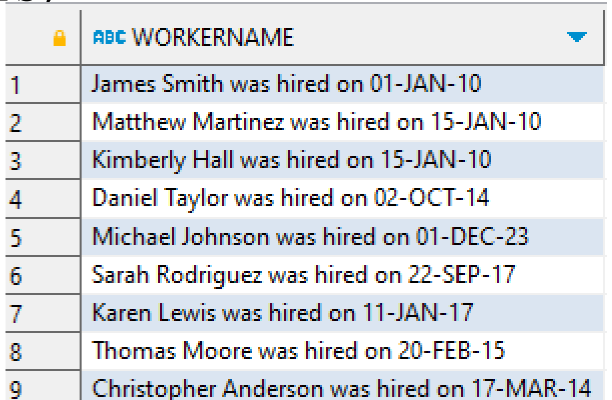


	WORKERNAME
1	James Smith works here!
2	Matthew Martinez works here!
3	Kimberly Hall works here!
4	Daniel Taylor works here!
5	Michael Johnson works here!
6	Sarah Rodriguez works here!
7	Karen Lewis works here!
8	Thomas Moore works here!
9	Christopher Anderson works here!

```
SELECT name || ' was hired on ' || hiredate as WorkerName FROM workers;
```



	EMPID	NAME	PHONE	TITLE	SALARY	HIREDATE
1	101	James Smith	2815550101	Owner	250,000	2010-01-01 00:00:00.000
2	102	Matthew Martinez	2815551919	Director of Facilities	125,000	2010-01-15 00:00:00.000
3	103	Kimberly Hall	7135551818	Director of Racing	160,000	2010-01-15 00:00:00.000
4	402	Daniel Taylor	2815551515	Director of Education	140,000	2014-10-02 00:00:00.000
5	414	Michael Johnson	7135550202	Race Coordinator	82,000	2023-12-01 00:00:00.000
6	436	Sarah Rodriguez	7135551010	Ranch Hand	55,000	2017-09-22 00:00:00.000
7	557	Karen Lewis	2815551111	Race Coordinator	78,000	2017-01-11 00:00:00.000
8	599	Thomas Moore	7135551414	Health and Nutrition Specialist	135,000	2015-02-20 00:00:00.000
9	670	Christopher Anderson	7135551616	Ranch Hand	52,000	2014-03-17 00:00:00.000



	WORKERNAME
1	James Smith was hired on 01-JAN-10
2	Matthew Martinez was hired on 15-JAN-10
3	Kimberly Hall was hired on 15-JAN-10
4	Daniel Taylor was hired on 02-OCT-14
5	Michael Johnson was hired on 01-DEC-23
6	Sarah Rodriguez was hired on 22-SEP-17
7	Karen Lewis was hired on 11-JAN-17
8	Thomas Moore was hired on 20-FEB-15
9	Christopher Anderson was hired on 17-MAR-14

The SUBSTR Function

- SUBSTR (char, m [,n]) returns a portion of char, beginning at character m, n characters long (if n is omitted, to the end of char). The first position of char is 1.

- Example:
  - `SELECT SUBSTR('ABCDEFG',3,4) AS "Substring" FROM DUAL;`
  - Returns: CDEF

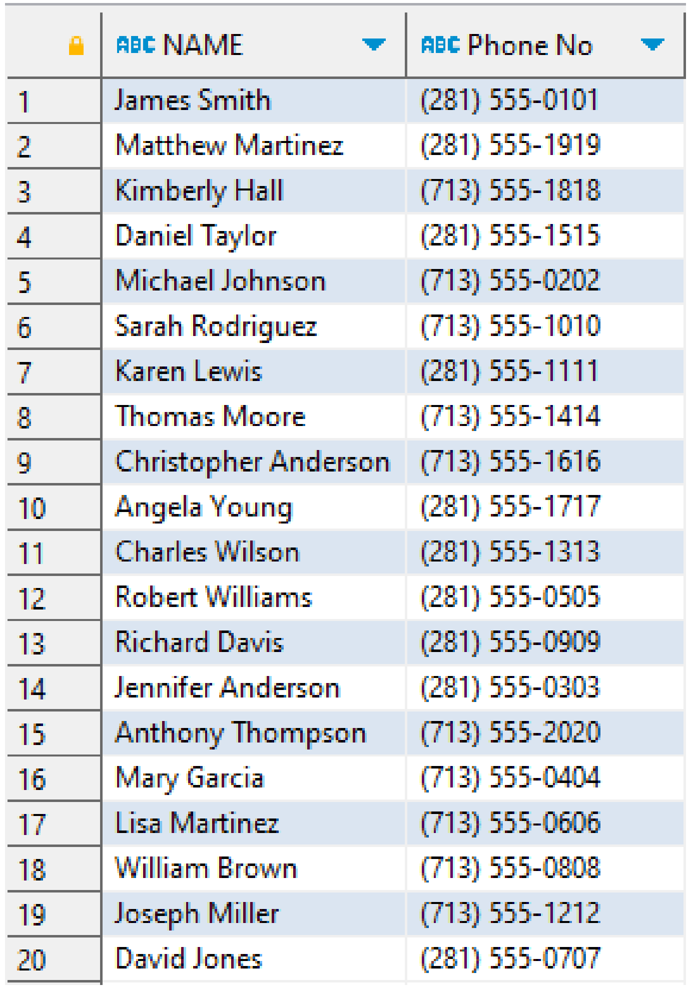
- The SUBSTR function is often used in conjunction with the concatenation operator (||).

Use of the Concatenation Operator (||)

- Display the name and phone number of all workers with phone formatted as (xxx)xxx-xxxx.

-
  ```
  SELECT name, '(' ||
  SUBSTR(phone,1,3) || ') ' ||
  SUBSTR(phone,4,3) || '-' ||
  SUBSTR(phone,7,4) AS "Phone No"
  FROM workers;
  ```

- We wouldn’t be able to do these string manipulations (like substr) if we had stored phone as a numeric attribute!



	NAME	Phone No
1	James Smith	(281) 555-0101
2	Matthew Martinez	(281) 555-1919
3	Kimberly Hall	(713) 555-1818
4	Daniel Taylor	(281) 555-1515
5	Michael Johnson	(713) 555-0202
6	Sarah Rodriguez	(713) 555-1010
7	Karen Lewis	(281) 555-1111
8	Thomas Moore	(713) 555-1414
9	Christopher Anderson	(713) 555-1616
10	Angela Young	(281) 555-1717
11	Charles Wilson	(281) 555-1313
12	Robert Williams	(281) 555-0505
13	Richard Davis	(281) 555-0909
14	Jennifer Anderson	(281) 555-0303
15	Anthony Thompson	(713) 555-2020
16	Mary Garcia	(713) 555-0404
17	Lisa Martinez	(713) 555-0606
18	William Brown	(713) 555-0808
19	Joseph Miller	(713) 555-1212
20	David Jones	(281) 555-0707

The INSTR Function

- The INSTR function is used to return the numeric value of the location of a character string within a character column or character literal

- Syntax: INSTR (char1, char2 [,n[,m]])

- Its purpose is to locate the position of the mth occurrence of char2 in char1, beginning the search at position n.
    - If m is omitted, 1 is assumed.
    - If n is omitted, 1 is assumed.
    - The position is given relative to the first character of char1, even when n > 1.

INSTR Function Examples

- Example:

```
              12345678901
SELECT INSTR('MISSISSIPPI','S',5,2) AS "In String Example" FROM DUAL;
```

- Returns: 7

- Example:

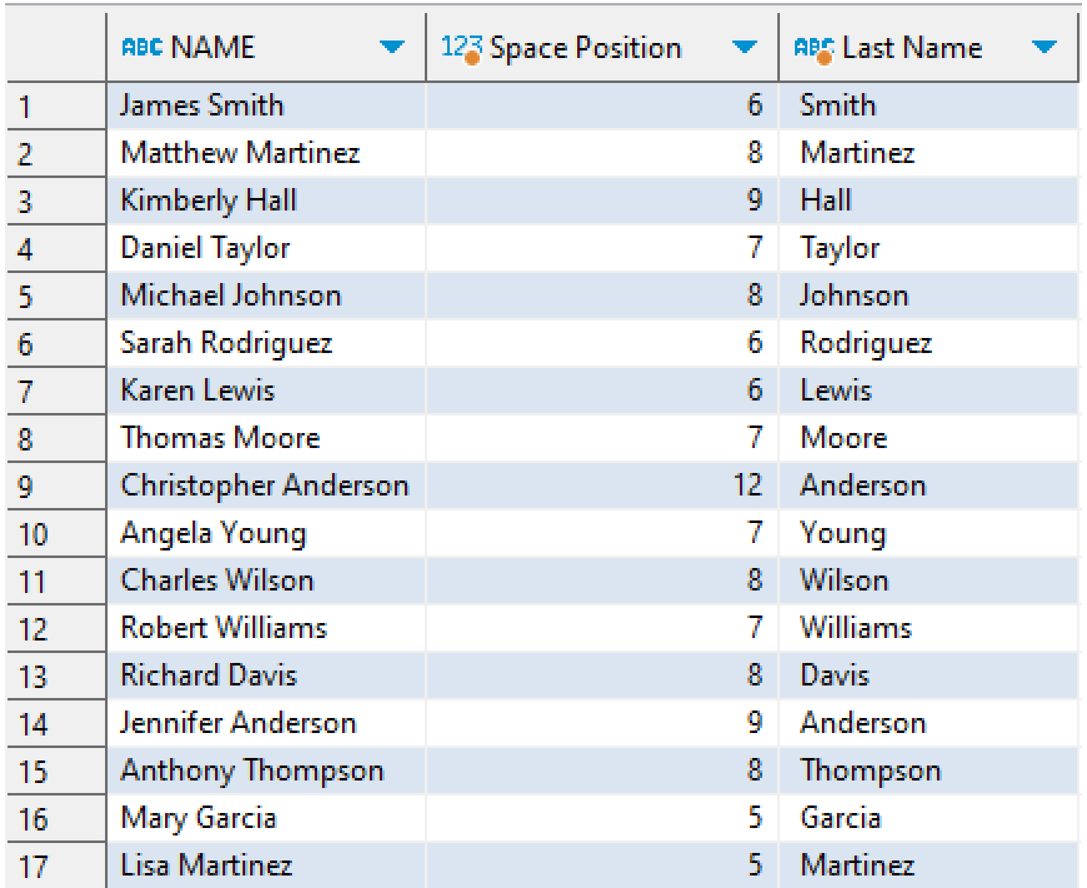
```
              12345678901
SELECT INSTR('MISSISSIPPI','S',5,1) AS "In String Example" FROM DUAL;
```

- Returns: 6

Use of INSTR and SUBSTR Functions

```
SELECT name,
instr(name,  ' ') AS "Space Position",
substr(name, instr(name, ' ')) AS "Last Name"
FROM workers;
```

Extracting the last name →

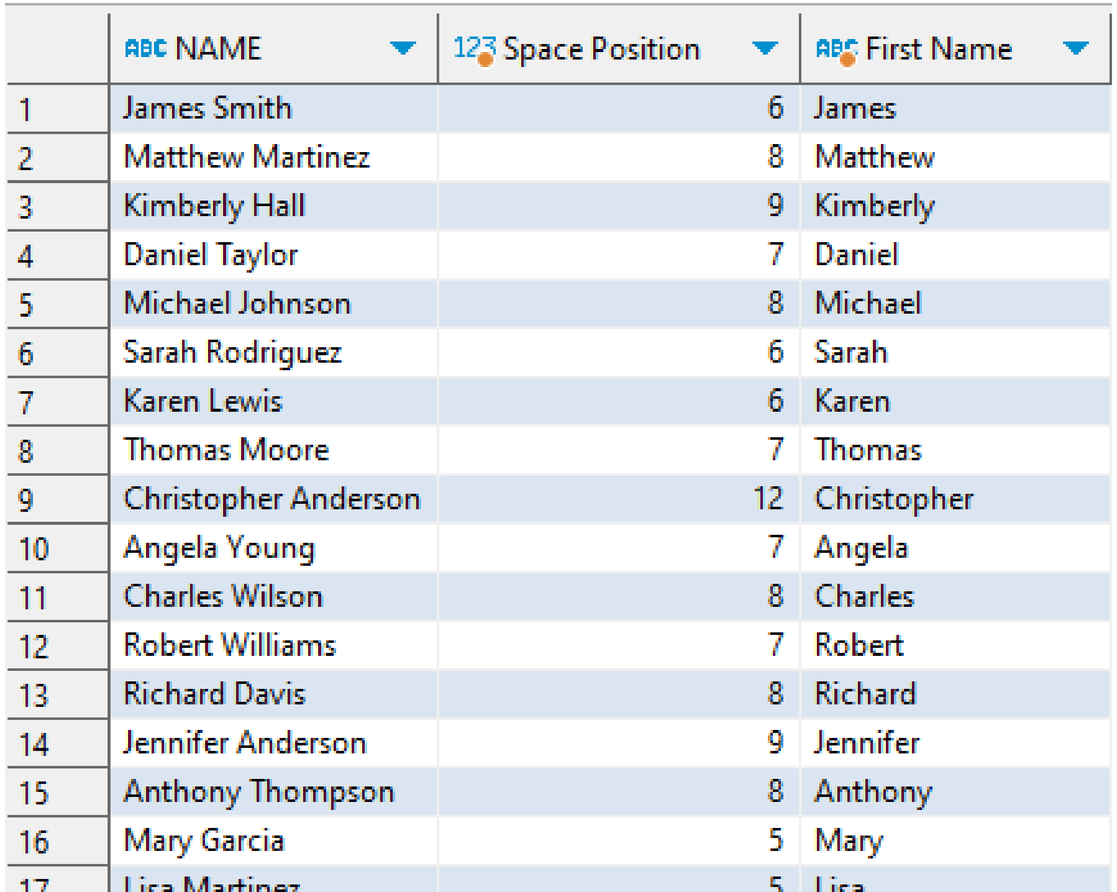


	NAME	Space Position	Last Name
1	James Smith	6	Smith
2	Matthew Martinez	8	Martinez
3	Kimberly Hall	9	Hall
4	Daniel Taylor	7	Taylor
5	Michael Johnson	8	Johnson
6	Sarah Rodriguez	6	Rodriguez
7	Karen Lewis	6	Lewis
8	Thomas Moore	7	Moore
9	Christopher Anderson	12	Anderson
10	Angela Young	7	Young
11	Charles Wilson	8	Wilson
12	Robert Williams	7	Williams
13	Richard Davis	8	Davis
14	Jennifer Anderson	9	Anderson
15	Anthony Thompson	8	Thompson
16	Mary Garcia	5	Garcia
17	Lisa Martinez	5	Martinez

Use of INSTR and SUBSTR Functions

```
SELECT name,
instr(name,  ' ') AS "Space Position",
substr(name, 1, instr(name, ' ')) AS "First Name"
FROM workers;
```

Extracting the first name →



	NAME	Space Position	First Name
1	James Smith	6	James
2	Matthew Martinez	8	Matthew
3	Kimberly Hall	9	Kimberly
4	Daniel Taylor	7	Daniel
5	Michael Johnson	8	Michael
6	Sarah Rodriguez	6	Sarah
7	Karen Lewis	6	Karen
8	Thomas Moore	7	Thomas
9	Christopher Anderson	12	Christopher
10	Angela Young	7	Angela
11	Charles Wilson	8	Charles
12	Robert Williams	7	Robert
13	Richard Davis	8	Richard
14	Jennifer Anderson	9	Jennifer
15	Anthony Thompson	8	Anthony
16	Mary Garcia	5	Mary
17	Lisa Martinez	5	Lisa

The LENGTH Function

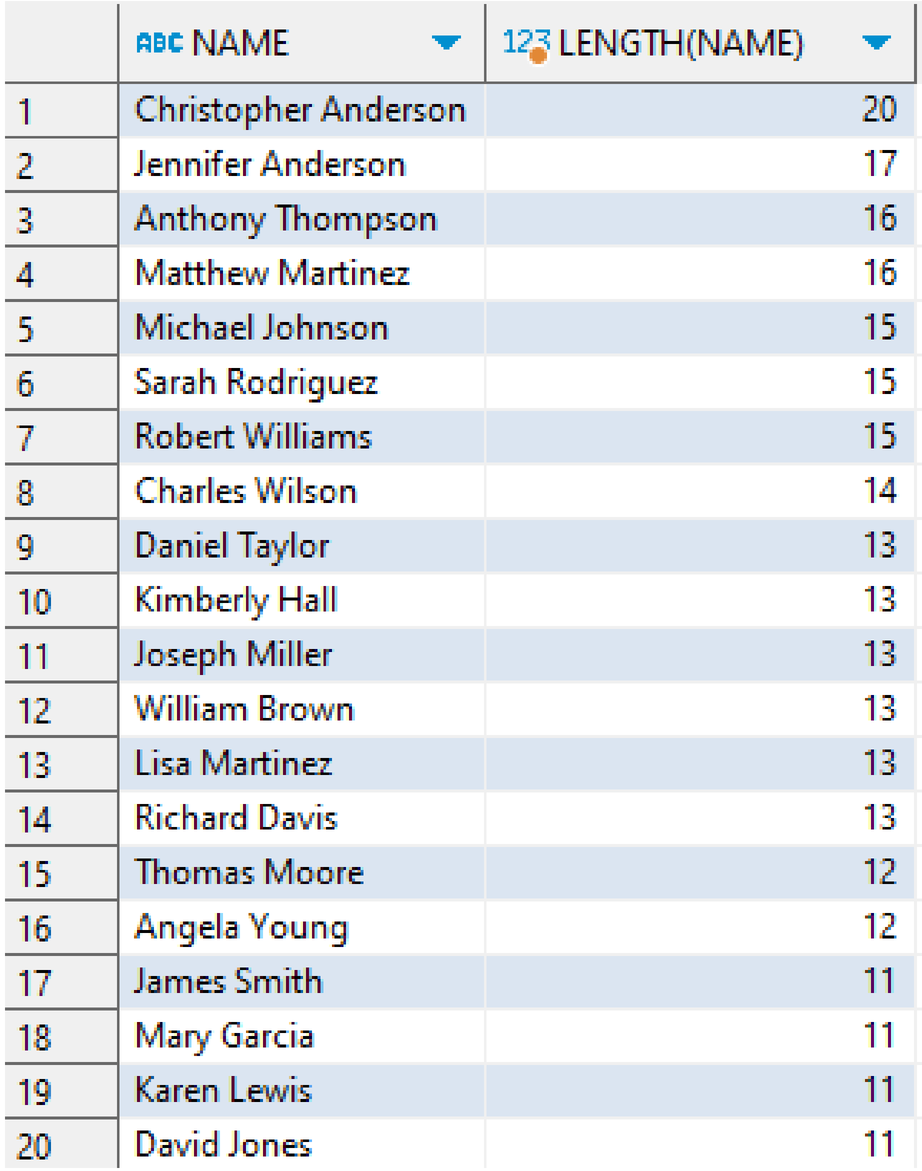
- The LENGTH (char) function returns the length of the character string char.
- Example:

  ```
  SELECT LENGTH('Jones, John') FROM DUAL;
  ```

  Returns: 11
- Note: attributes of datatype char (as opposed to varchar) return a length that includes all trailing blank spaces

The LENGTH Function

```
SELECT name, LENGTH(name)
FROM workers
ORDER BY LENGTH(name) DESC;
```



	NAME	LENGTH(NAME)
1	Christopher Anderson	20
2	Jennifer Anderson	17
3	Anthony Thompson	16
4	Matthew Martinez	16
5	Michael Johnson	15
6	Sarah Rodriguez	15
7	Robert Williams	15
8	Charles Wilson	14
9	Daniel Taylor	13
10	Kimberly Hall	13
11	Joseph Miller	13
12	William Brown	13
13	Lisa Martinez	13
14	Richard Davis	13
15	Thomas Moore	12
16	Angela Young	12
17	James Smith	11
18	Mary Garcia	11
19	Karen Lewis	11
20	David Jones	11

The RTRIM Function

- The RTRIM (char [, set]) function returns char, with final characters removed after the last character not in set.

- If no set of characters is specified, set defaults to `' '` (a blank space) and the function trims off trailing blanks.

- The RTRIM function operates on the rightmost characters in a string in the same way that the LTRIM function operates on the leftmost characters in a string.

- Example:
  - `SELECT RTRIM('STINSONxxXxx','x') AS "Right Trim Example" FROM DUAL;`
  - Returns: STINSONxxX

  - `SELECT RTRIM('Houston         ') AS "Right Trim Example" FROM DUAL;`
  - Returns: Houston

- Note: char (as opposed to varchar) deliver different results since char data type contains embedded trailing blanks.

The LTRIM Function

- The LTRIM (char [, set]) function removes unwanted characters from the left of char, with initial characters removed up to the first character not in set.

- If no set of characters is specified, set defaults to `' '` (a blank space) and the function trims off leading blank spaces.

- Example:
  - `SELECT LTRIM('xxxXxxLAST WORD', 'x') AS "Left Trim Example" FROM DUAL;`
  - Returns: XxxLASTWORD

- Note: LTRIM is case-sensitive

The LPAD and RPAD Functions

- The LPAD and RPAD functions allow you to “pad” the left (and right) side of a column or character string with a set of characters.

- Syntax: LPAD/RPAD (string, length [,'set'])
  - string is the name of the character column (or a literal string),
  - length is the total number of characters long that the result should be (i.e., its width), and
  - set is the set of characters that do the padding

LPAD and RPAD Examples

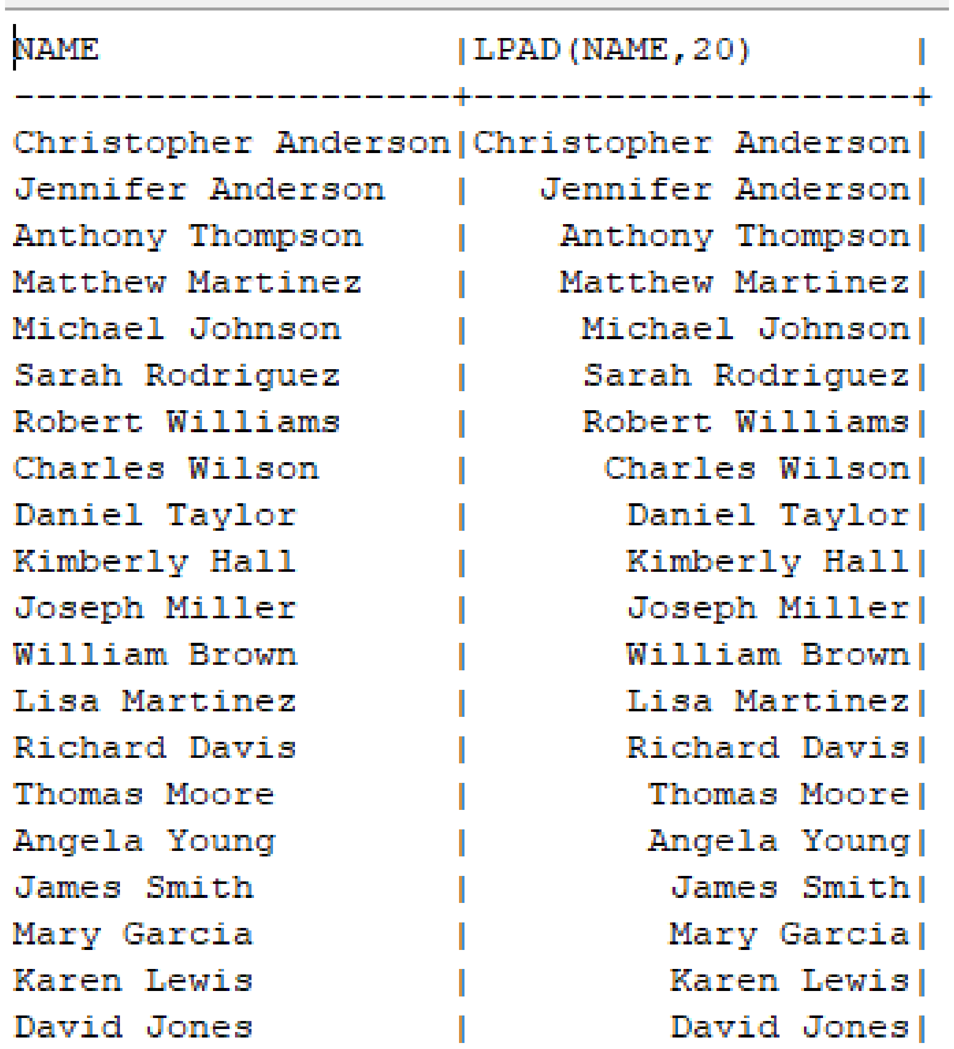
- `SELECT LPAD('Page 1', 14, '*') AS "LPAD Example" FROM DUAL;`
  - Returns: ********Page 1

- `SELECT RPAD('Page 1', 14, '*.') AS "RPAD Example" FROM DUAL;`
  - Returns: Page 1*.*.*.*.

LPAD Example

```
SELECT name, lpad(name,20)
FROM workers
ORDER BY LENGTH(name) DESC;
```

(The impact of this is a little easier to see in the text view in Dbeaver, since it uses a fixed width font…)→

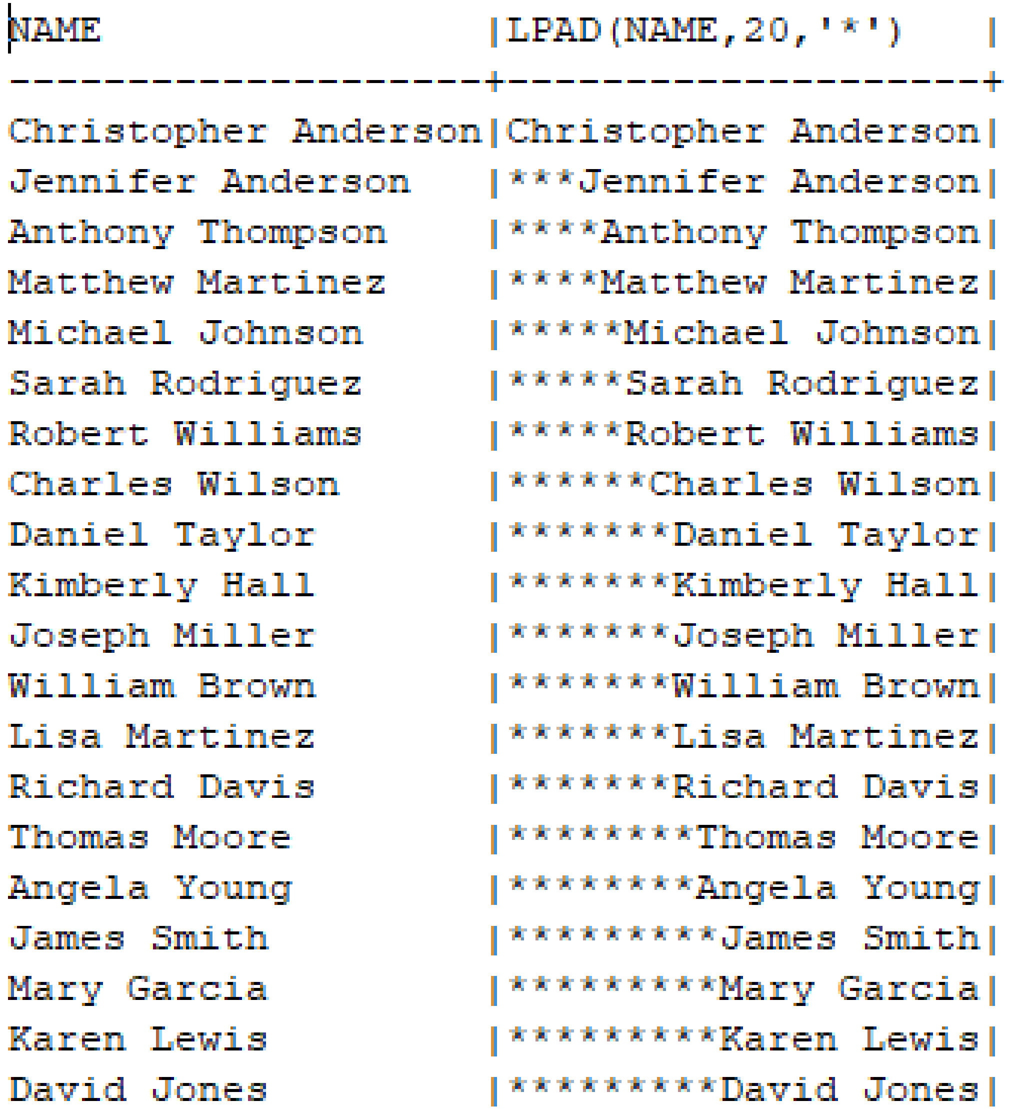


```
NAME                |LPAD(NAME,20)       |
--------------------+--------------------+
Christopher Anderson|Christopher Anderson|
Jennifer Anderson   |   Jennifer Anderson|
Anthony Thompson    |    Anthony Thompson|
Matthew Martinez    |    Matthew Martinez|
Michael Johnson     |     Michael Johnson|
Sarah Rodriguez     |     Sarah Rodriguez|
Robert Williams     |     Robert Williams|
Charles Wilson      |      Charles Wilson|
Daniel Taylor       |       Daniel Taylor|
Kimberly Hall       |       Kimberly Hall|
Joseph Miller       |       Joseph Miller|
William Brown       |       William Brown|
Lisa Martinez       |       Lisa Martinez|
Richard Davis       |       Richard Davis|
Thomas Moore        |        Thomas Moore|
Angela Young        |        Angela Young|
James Smith         |         James Smith|
Mary Garcia         |         Mary Garcia|
Karen Lewis         |         Karen Lewis|
David Jones         |         David Jones|
```

LPAD and Example

```
SELECT name, lpad(name,20,'*')
FROM workers
ORDER BY LENGTH(name) DESC;
```

(The impact of this is a little easier to see in the text view in Dbeaver, since it uses a fixed width font…)→



```
NAME                |LPAD(NAME,20,'*')   |
--------------------+--------------------+
Christopher Anderson|Christopher Anderson|
Jennifer Anderson   |***Jennifer Anderson|
Anthony Thompson    |****Anthony Thompson|
Matthew Martinez    |****Matthew Martinez|
Michael Johnson     |*****Michael Johnson|
Sarah Rodriguez     |*****Sarah Rodriguez|
Robert Williams     |*****Robert Williams|
Charles Wilson      |******Charles Wilson|
Daniel Taylor       |*******Daniel Taylor|
Kimberly Hall       |*******Kimberly Hall|
Joseph Miller       |*******Joseph Miller|
William Brown       |*******William Brown|
Lisa Martinez       |*******Lisa Martinez|
Richard Davis       |*******Richard Davis|
Thomas Moore        |********Thomas Moore|
Angela Young        |********Angela Young|
James Smith         |*********James Smith|
Mary Garcia         |*********Mary Garcia|
Karen Lewis         |*********Karen Lewis|
David Jones         |*********David Jones|
```

The DECODE Function

- The DECODE (value, search_value, result, default_value) function is used to compare value with search_value. If the values are equal, the DECODE function returns result; otherwise, default_value is returned. The DECODE function allows you to perform if-then-else logic in SQL within a row.

- Example: Note how the DECODE Function allows students to be listed in descending order by grade level (GR, SR, …, FR)

-
  ```
  SELECT SID, NAME, GRADELEVEL
  FROM STUDENT
  ORDER BY DECODE (GRADELEVEL, 'FR', '1', 'SO', '2', 'JR', '3', 'SR',
  4, 'GR', 5) DESC;
  ```

Case Expression in SQL

- The Oracle/PLSQL CASE expression has the functionality of an IF-THEN-ELSE statement. You can use the CASE expression within a SQL statement.

- Searched Case Expression Format

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    …
    WHEN conditionN THEN result
    ELSE default_result
END
```

- where condition1, condition2, …, conditionN are the conditions to be evaluated
- result1, result2, …, resultN are the returned results (one for each possible condition)
- default_result is the default result returned when no true condition is found

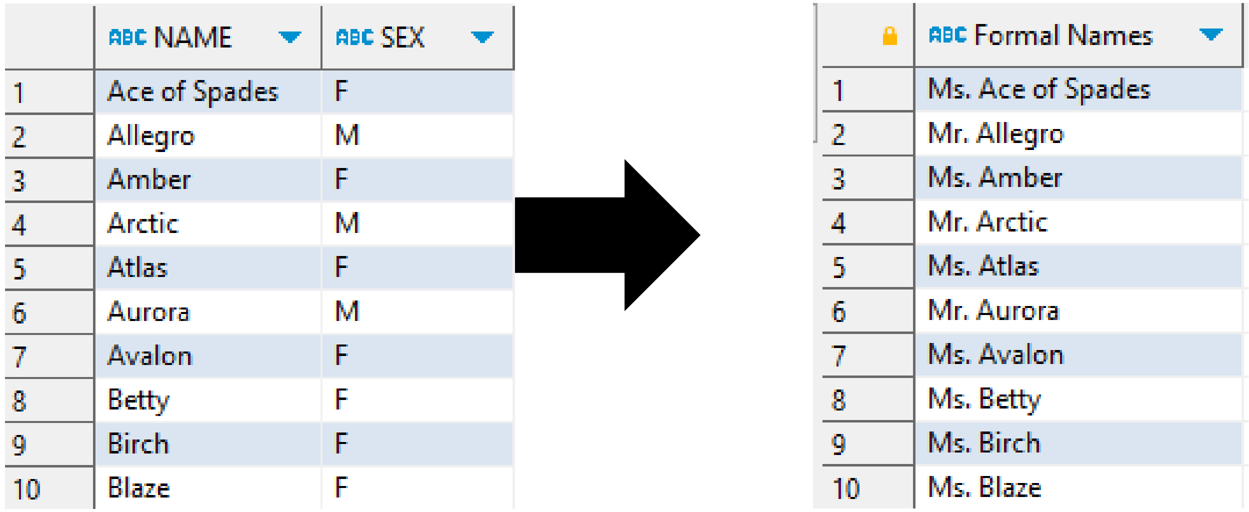
Case Expression in SQL

- Can be used in a similar manner to decode:

```
SELECT SID, NAME, GRADELEVEL FROM STUDENT ORDER BY CASE
WHEN gradelevel='FR' THEN '1', WHEN gradelevel='SO' THEN '2',
WHEN gradelevel='JR' THEN '3', WHEN gradelevel='SR' THEN '4',
WHEN gradelevel='GR' THEN '5' END DESC;
```

- Or to replace values inline – for example, you could do something like:

  -
    ```
    SELECT CASE WHEN Sex='M' then 'Mr. ' ELSE 'Ms. ' END || name AS "Formal Names"
    FROM horses;
    ```



	NAME	SEX
1	Ace of Spades	F
2	Allegro	M
3	Amber	F
4	Arctic	M
5	Atlas	F
6	Aurora	M
7	Avalon	F
8	Betty	F
9	Birch	F
10	Blaze	F

	Formal Names
1	Ms. Ace of Spades
2	Mr. Allegro
3	Ms. Amber
4	Mr. Arctic
5	Ms. Atlas
6	Mr. Aurora
7	Ms. Avalon
8	Ms. Betty
9	Ms. Birch
10	Ms. Blaze

Review: Single-Row Character Functions

- SUBSTR(char, m [,n])
  - Returns the portion of *char* starting at *m* and continuing for *n* characters

- LENGTH(char)
  - Returns the number of characters in *char*

- LTRIM(char [, set]) and RTRIM(char [, 'set'])
  - Removes characters in *set* from the left or right of *char* (default is space)

- LPAD/RPAD(char, length [,'set'])
  - Adds *length* characters in *set* to the left or right of *char* (default is space)

- INSTR (char1, char2 [,n[,m]])
  - Returns the position of the m^{th} occurrence of *char2* in *char1*, starting at position *n*, by default, the first occurrence starting at position 1

Module 13.1

String manipulation SQL functions

- Concatenation
- SUBSTRING
- LENGTH
- TRIM (RTRIM / LTRIM)
- PAD (RPAD / LPAD)
- INSTR
- DECODE
- CASE

Module 13.2

Date and Time

- TO_CHAR format mask
- TO_DATE format mask

Dates and Times

- Least standardized data type across platforms

- Oracle 10g displays date values in a default DD-MON-YY format that represents a two-digit day, a three-letter month abbreviation, and a two-digit year (e.g., the date April 22, 2007 would be represented as 22-APR-07).

- Although referenced as a non-numeric field, a date is actually stored internally in a numeric format.

- Uses the Julian calendar (e.g., 2454213 = April 22, 2007)

The TO_CHAR and TO_Date Functions

- TO_CHAR function is used to extract the different parts of a date/time and convert them to a character string.

- TO_DATE function is used to convert character strings to a valid date format.

- Both functions use a format mask

- SYSDATE and CURRENT_DATE both return the current date and time
  - `SELECT SYSDATE FROM dual;`
  - `SELECT CURRENT_DATE FROM dual;`

The TO_CHAR and TO_Date Functions

- Using TO_CHAR masks

- To get the current month:
  - `select to_char(current_date, 'MON') from dual;`
  - `Returns: APR`

- To format a number as money:
  - `SELECT to_char('50000','$99,999') from dual;`
  - `Returns: $50,000`

  - `SELECT to_char('8000','$99,999') from dual;`
  - `Returns: $8,000`

- To convert a string to a date:
  - `SELECT to_date('22-MAR-2018') from dual;`

- Try this in the SQL project to get more details about a flight time:
  - `SELECT FL_Flight_no, FL_orig, FL_orig_time, TO_CHAR(FL_Orig_time, 'MM-DD-YYYY HH:MI PM') FROM Flight;`

Selected Date and Time Format Elements Used With the TO_CHAR and TO_DATE Functions

Element	Description	Example
MONTH, Month, or month	Name of the month spelled out (padded with blank spaces to a total width of nine spaces); case follows format.	JULY, July, or july (5 spaces follows each representation of July)
MON, Mon, or mon	Three-letter abbreviation of the name of the month; case follows format.	JUL, Jul, or jul
MM	Two-digit numeric value of the month	7
D	Numeric value of the day of the week	Monday = 2
DD	Numeric value of the day of the month	23
DAY, Day, or day	Name of the day of the week spelled out (padded with blank spaces to a length of nine characters)	MONDAY, Monday, or monday (3 spaces follows each representation of Monday)
fm	“Fill mode.” When this element appears, subsequent elements (such as MONTH) suppress blank padding, leaving a variable-length result.	fmMonth, yyyy produces a date such as March, 2014
DY	Three-letter abbreviation of the day of the week	MON, Mon, or mon
YYYY	Four-digit year	2014
YY	Last two digits of the year	14
YEAR, Year, or year	Spells out the year; case follows year.	TWO THOUSAND FOURTEEN
BC or AD	Indicates B.C. or A.D.	2014 A.D.
AM or PM	Meridian indicator	10:00 AM
J	Julian date; January 1, 4712 B.C. is day 1.	July 27, 2014 is Julian date 2456865
SS	Seconds (value between 0 and 59)	21
MI	Minutes (value between 0 and 59)	32
HH	Hours (value between 1 and 12)	9
HH24	Hours (value between 0 and 23)	13

Selected Number Format Elements Used With the TO_CHAR Function

Element	Description	Example
9	Series of 9s indicates width of display (with insignificant leading zeros not displayed).	99999
0	Displays insignificant leading zeros	0009999
$	Displays a floating dollar sign to prefix value	$99999
.	Indicates number of decimals to display	999.99
,	Displays a comma in the position indicated	9,999

Module 13.2

Date and Time

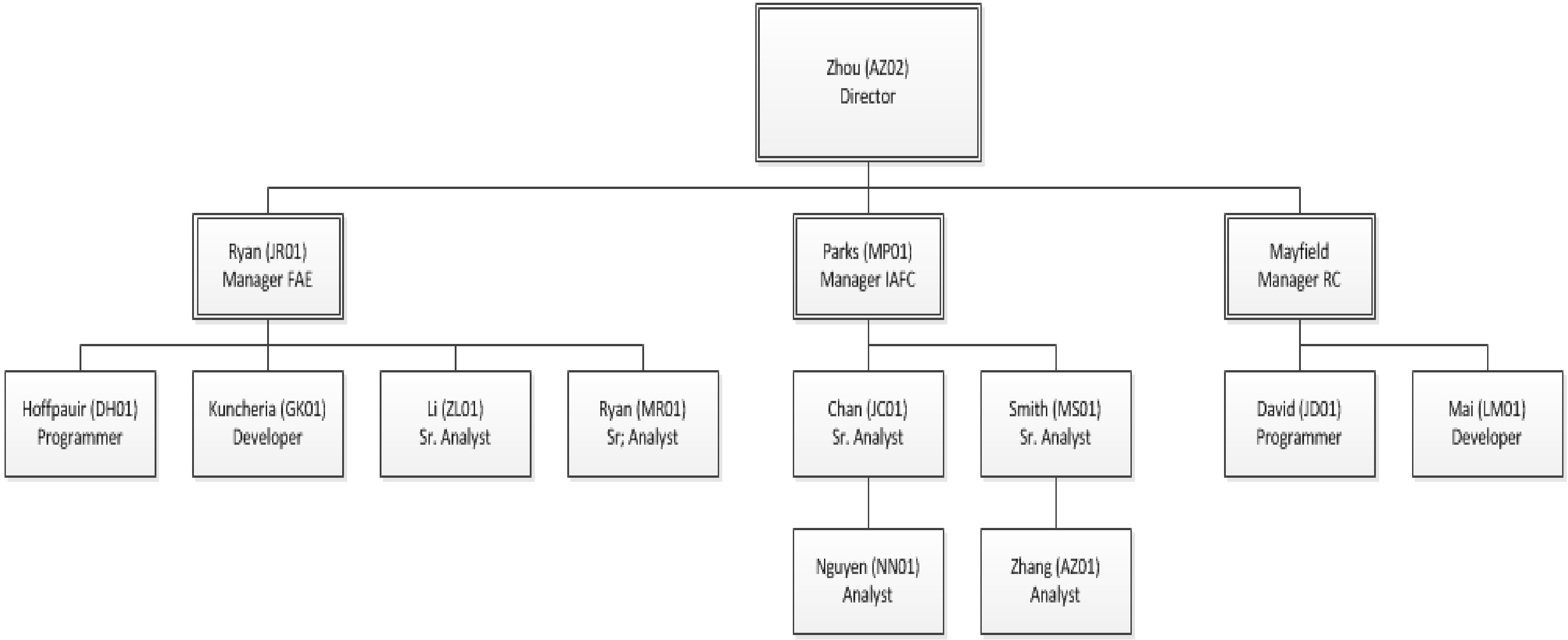
- TO_CHAR format mask
- TO_DATE format mask

Module 13.3

Hierarchical Queries

- START WITH
- CONNECT BY
- PRIOR
- LEVEL

Hierarchical Queries in SQL



Zhou (AZ02)
Director

Ryan (JR01)
Manager FAE

Parks (MP01)
Manager IAFC

Mayfield
Manager RC

Hoffpauir (DH01)
Programmer

Kuncheria (GK01)
Developer

Li (ZL01)
Sr. Analyst

Ryan (MR01)
Sr; Analyst

Chan (JC01)
Sr. Analyst

Smith (MS01)
Sr. Analyst

David (JD01)
Programmer

Mai (LM01)
Developer

Nguyen (NN01)
Analyst

Zhang (AZ01)
Analyst

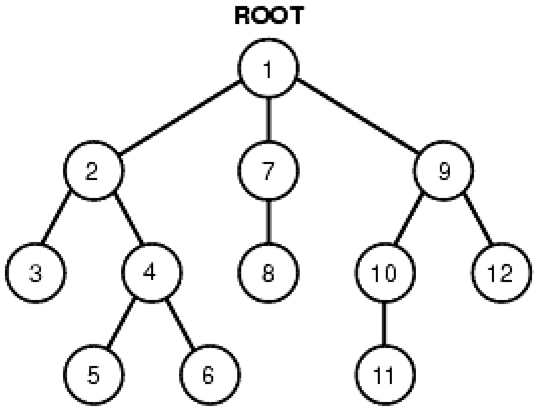
Start With – Connect By [Prior]

- Start With – indicates root node.

- Connect by – indicates relationships between parent and child rows within the hierarchy.

- Prior - keyword indicates the parent.

- LEVEL – pseudocolumn that returns
  - 1 for root
  - 2 for children of root
  - 3 for next child level, etc.



Level 1

Level 2

Level 3

Level 4

Start With – Connect By [Prior]

- The **Start With** clause identifies the row(s) to be used as the root(s) of a hierarchical query.
    - The clause specifies a condition that the roots must satisfy
    - If this clause is omitted, SQL uses all rows in the table as root
    - A Start With condition can contain a subquery, whereas a Connect By condition cannot contain a subquery.

- The **Connect By** clause specifies a relationship (condition) between parent and child rows in a hierarchical query.

- Some part of the condition must use the Prior operator to refer to the parent row. The part of the condition containing the Prior operator must have one of the following forms:
    - PRIOR *expr* <operator> *expr* **or**
      expr <operator> PRIOR *expr*

Start With and Connect By: Example 1

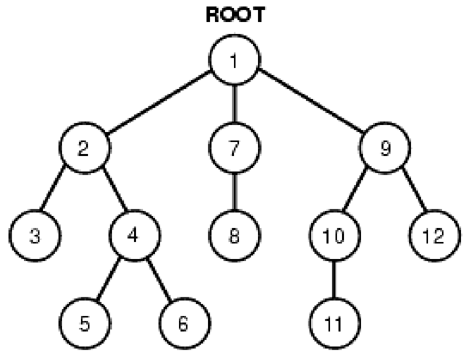
```
SELECT * FROM CONSULTANT
START WITH ID = 'MP01'
CONNECT BY PRIOR ID = REPTS_TO;
```

```
ID    NAME            GENDER TITLE        DID       SALARY HIREDATE  REPTS_TO
----- --------------- ------ ------------ ----- ---------- --------- ----------
MP01  Parks, Michael  M      Manager      IAFC      140000 09-DEC-08 AZ02
JC01  Chan, Jackie    M      Sr. Analyst  IAFC       80000           MP01
NN01  Nguyen, Nicole  F      Analyst      IAFC       42000 22-MAR-12 JC01
MS01  Smith, Maranda  F      Sr. Analyst  IAFC       72500 15-MAY-12 MP01
AZ01  Zhang, Anthony  M      Analyst      IAFC       65000 25-NOV-09 MS01
```

This query selects all consultants who are children (i.e., report to) of a specific parent (i.e., MP01).

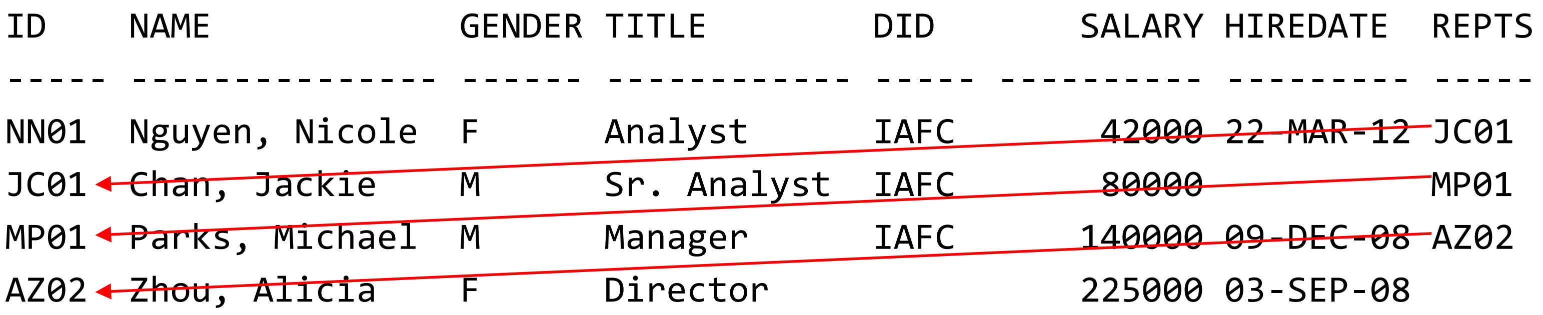
Starting at the top and working down –

Many possible paths/branches



Start With and Connect By: Example 2

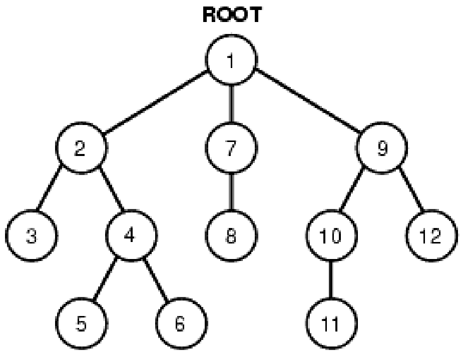
```
SELECT * FROM CONSULTANT
START WITH ID = 'NN01'
CONNECT BY ID = PRIOR REPTS_TO;
```



```
ID    NAME            GENDER TITLE        DID        SALARY HIREDATE  REPTS
----- --------------- ------ ------------ ----- ---------- --------- -----
NN01  Nguyen, Nicole  F      Analyst      IAFC        42000 22-MAR-12 JC01
JC01  Chan, Jackie    M      Sr. Analyst  IAFC        80000           MP01
MP01  Parks, Michael  M      Manager      IAFC       140000 09-DEC-08 AZ02
AZ02  Zhou, Alicia    F      Director                225000 03-SEP-08
```

This query selects all employees above the employee whose ID is NN01 in the hierarchy.

Starting at the bottom and working up –
Only one path to the top



Level Number

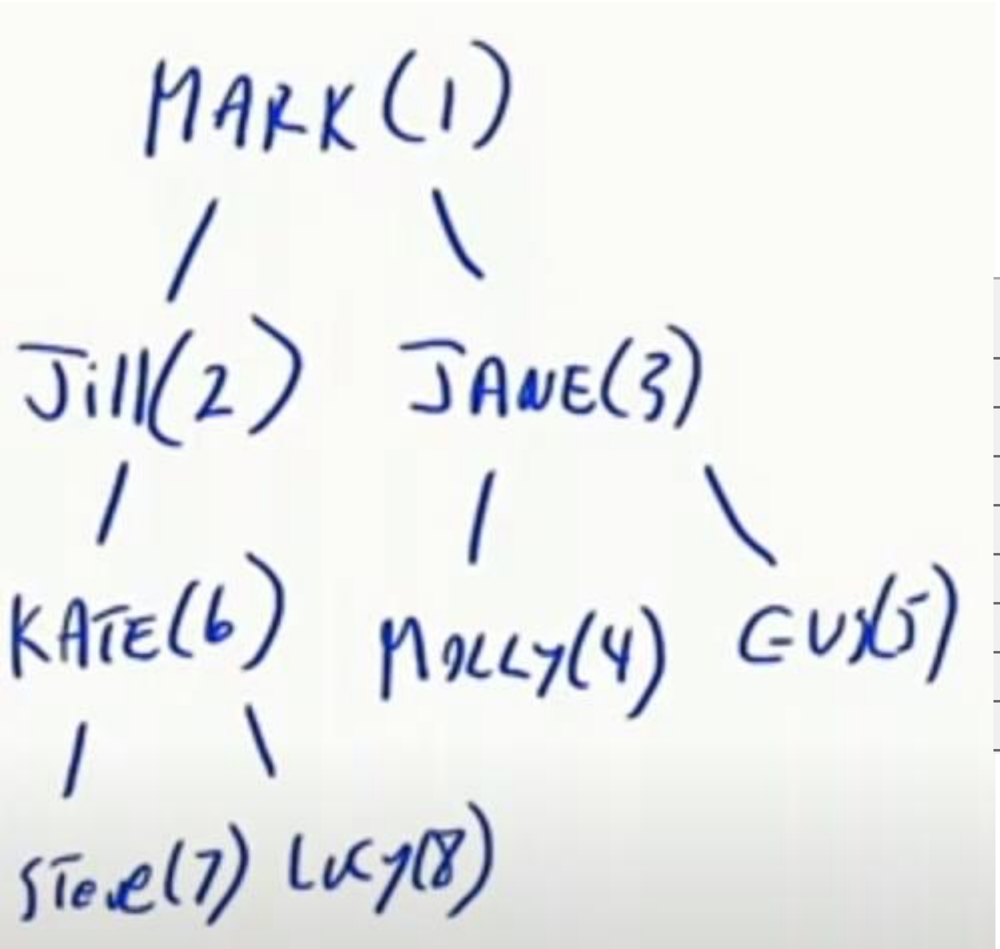
- Nodes in a tree are assigned level numbers, depending on how far removed they are from the root of the tree.
- LEVEL is a pseudo-column, which can be specified in an SQL statement where a column name may appear.

```
SELECT LEVEL, NAME, TITLE, SALARY FROM CONSULTANT
START WITH ID = 'AZ02'
CONNECT BY PRIOR ID = REPTS_TO

     LEVEL NAME            TITLE            SALARY
---------- --------------- ------------ ----------
         3 Hoffpauir, Deb  Programmer        80000
         3 Kuncheria, Ginu Developer         80000
         3 Chan, Jackie    Sr. Analyst       80000
         4 Nguyen, Nicole  Analyst           42000
         3 Smith, Maranda  Sr. Analyst       72500
         4 Zhang, Anthony  Analyst           65000
         3 David, Jason    Programmer        72500
         3 Mai, Ly H.      Developer         72000
```

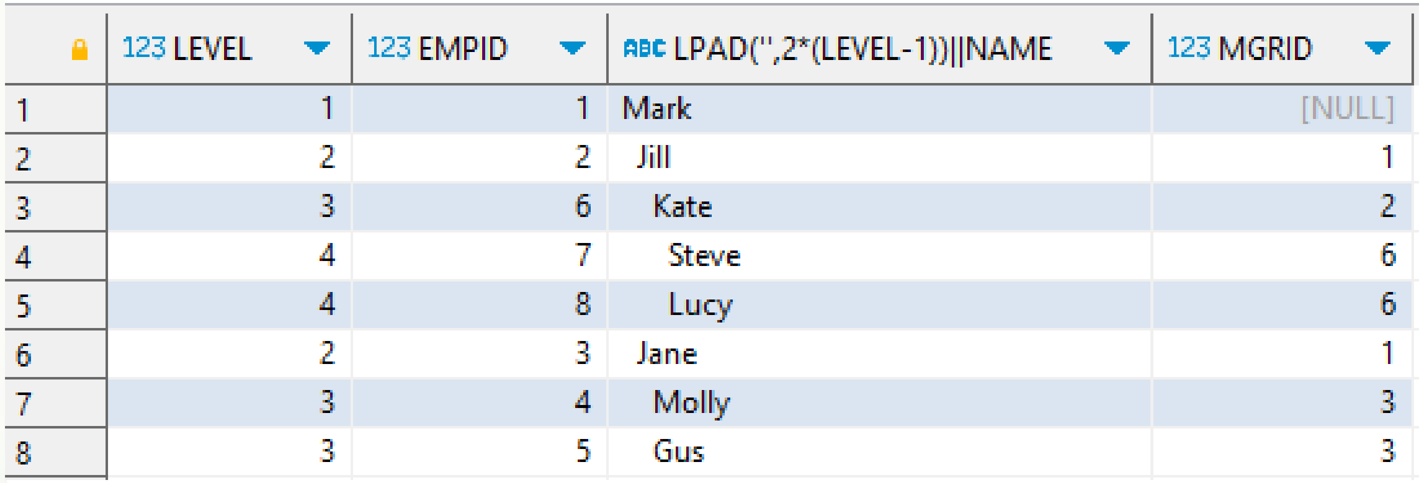
Combining Level Number With the LPAD Function To Pad a Consultant Name With Spaces

```
SELECT LEVEL, LPAD('  ', 2*(LEVEL-1))||NAME "Consultant Name"
        FROM CONSULTANT
        START WITH ID = 'AZ02'
        CONNECT BY PRIOR ID = REPTS_TO;

                    LEVEL Consultant Name
               ---------- -------------------------
                        1 Zhou, Alicia
                        2   Ryan, James R.
                        3     Hoffpauir, Deb
                        3     Kuncheria, Ginu
                        3     Ryan, Michael
                        3     Li, ZP
                        2   Parks, Michael
                        3     Chan, Jackie
                        4       Nguyen, Nicole
                        3     Smith, Maranda
                        4       Zhang, Anthony
                        2   Mayfield, Ron
                        3     David, Jason
                        3     Mai, Ly H.
```

A few weeks ago we did this:



```
SELECT level, empid,
LPAD('  ', 2*(LEVEL-1))||NAME, mgrid
FROM employee
START WITH empid = 1
CONNECT BY PRIOR empid = MGRID;
```



	123 LEVEL	123 EMPID	ABC LPAD('',2*(LEVEL-1))||NAME	123 MGRID
1	1	1	Mark	[NULL]
2	2	2	Jill	1
3	3	6	Kate	2
4	4	7	Steve	6
5	4	8	Lucy	6
6	2	3	Jane	1
7	3	4	Molly	3
8	3	5	Gus	3

Module 13.3

Hierarchical Queries

- START WITH
- CONNECT BY
- PRIOR
- LEVEL

Module 13.4

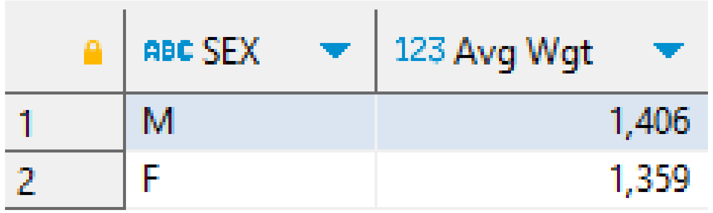
Extended GROUP BY clauses

- ROLLUP
- CUBE
- GROUPING
- GROUPING SETS

GROUP BY vs. ROLLUP

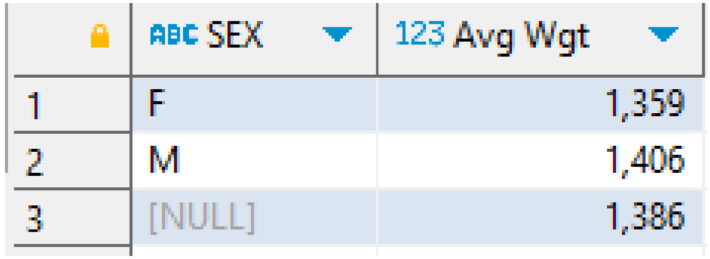
The ROLLUP Operator extends the GROUP BY clause to return a row containing a subtotal for each group along with a total for all groups.

```
SELECT sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY sex;
```



	SEX	Avg Wgt
1	M	1,406
2	F	1,359

```
SELECT sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY ROLLUP(sex);
```



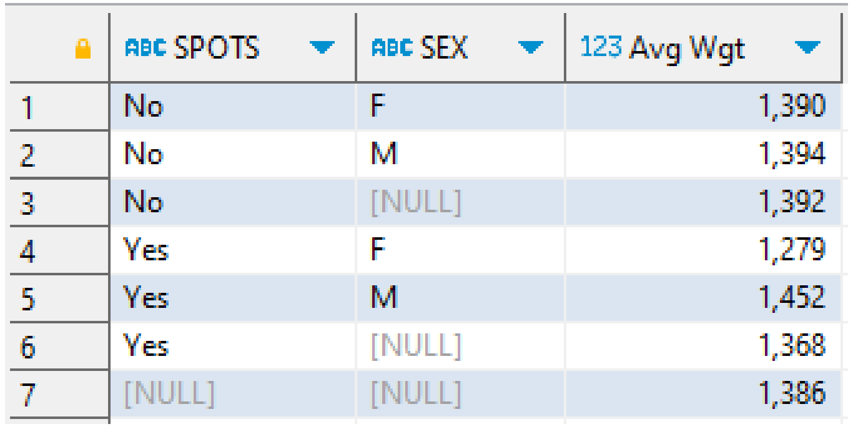
	SEX	Avg Wgt
1	F	1,359
2	M	1,406
3	[NULL]	1,386

Using Multiple Columns With ROLLUP

The ROLLUP Operator groups on several levels simultaneously. However, when you change the order of the group fields, you obtain a different sequence of groups. Observe the difference in the number of rows generated by the query in the two examples:

```
SELECT spots, sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY ROLLUP(spots, sex);
```

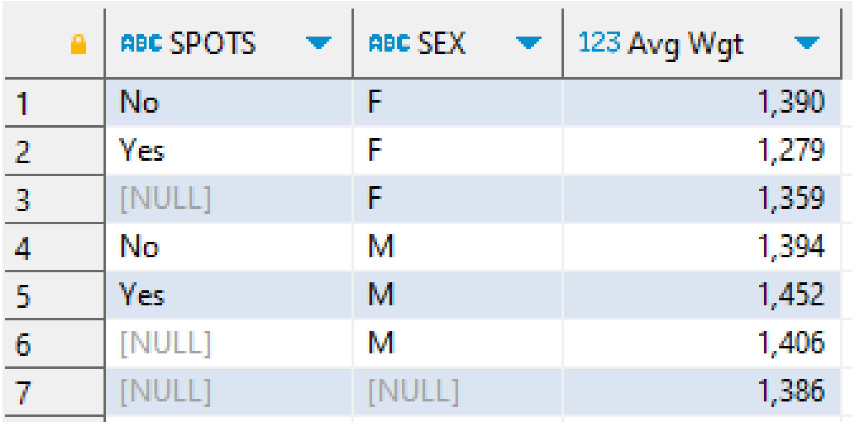
Avg for spots in the rollup →



	SPOTS	SEX	Avg Wgt
1	No	F	1,390
2	No	M	1,394
3	No	[NULL]	1,392
4	Yes	F	1,279
5	Yes	M	1,452
6	Yes	[NULL]	1,368
7	[NULL]	[NULL]	1,386

```
SELECT spots, sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY ROLLUP(sex, spots);
```

Avg for sex in the rollup →

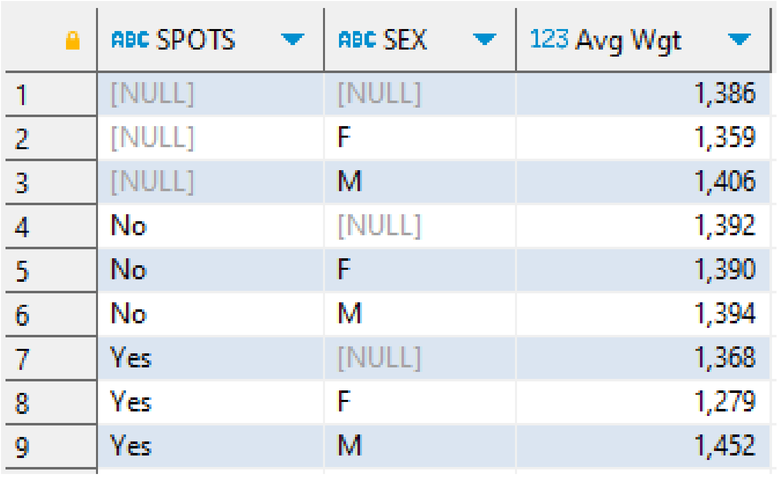


	SPOTS	SEX	Avg Wgt
1	No	F	1,390
2	Yes	F	1,279
3	[NULL]	F	1,359
4	No	M	1,394
5	Yes	M	1,452
6	[NULL]	M	1,406
7	[NULL]	[NULL]	1,386

The CUBE Operator

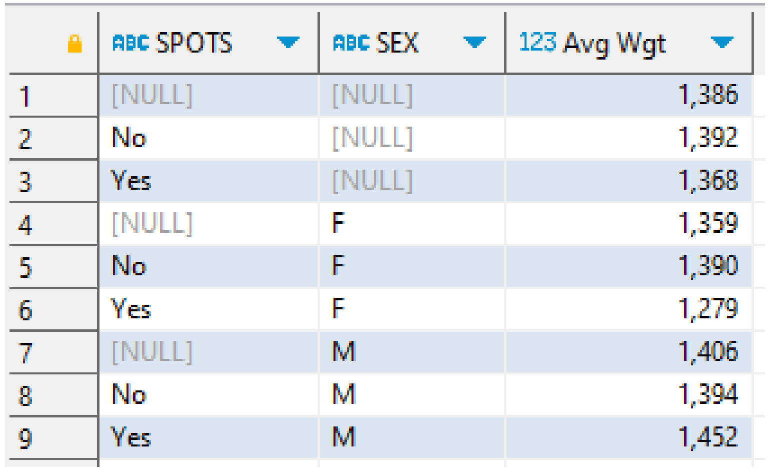
The CUBE Operator subtotals **all possible combinations** of the group by columns. Unlike the examples on the previous slides, we have all the same data in the examples below just in different orders.

```
SELECT spots, sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY CUBE(spots,sex);
```



	SPOTS	SEX	Avg Wgt
1	[NULL]	[NULL]	1,386
2	[NULL]	F	1,359
3	[NULL]	M	1,406
4	No	[NULL]	1,392
5	No	F	1,390
6	No	M	1,394
7	Yes	[NULL]	1,368
8	Yes	F	1,279
9	Yes	M	1,452

```
SELECT spots, sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY CUBE(sex, spots);
```

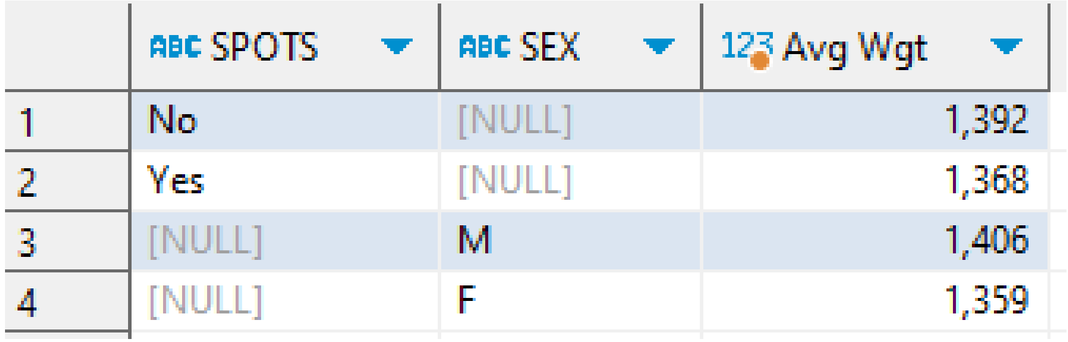


	SPOTS	SEX	Avg Wgt
1	[NULL]	[NULL]	1,386
2	No	[NULL]	1,392
3	Yes	[NULL]	1,368
4	[NULL]	F	1,359
5	No	F	1,390
6	Yes	F	1,279
7	[NULL]	M	1,406
8	No	M	1,394
9	Yes	M	1,452

The GROUPING SETS Extension

The GROUPING SETS extension to the GROUP BY clause is used to compute selective results for the set of groups you want to create. This example gets just the average weights by spots and sex.

```
SELECT spots, sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY GROUPING SETS(spots,sex);
```

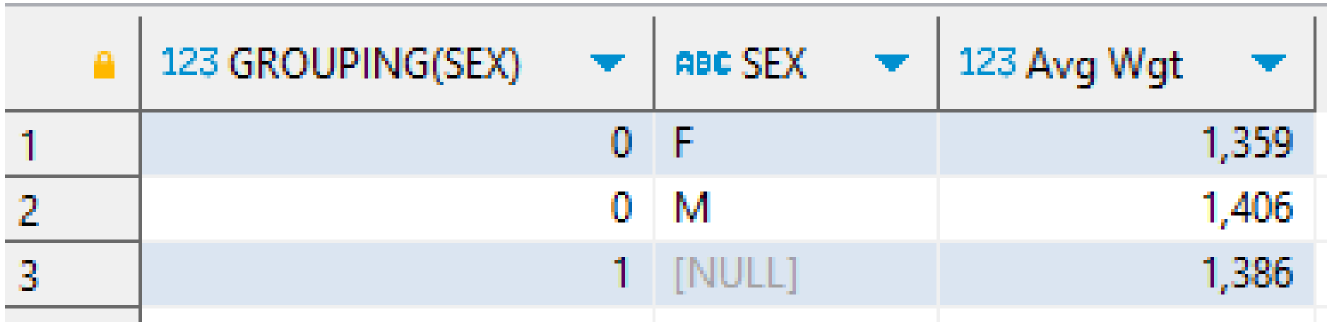


	ABC SPOTS	ABC SEX	123 Avg Wgt
1	No	[NULL]	1,392
2	Yes	[NULL]	1,368
3	[NULL]	M	1,406
4	[NULL]	F	1,359

The GROUPING (column) Function

The GROUPING (column) function accepts a column and returns a 0 or 1. A 1 is returned when the GROUPING function encounters a null value created by a ROLLUP or CUBE.

```
SELECT GROUPING(sex), sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY ROLLUP(sex);
```

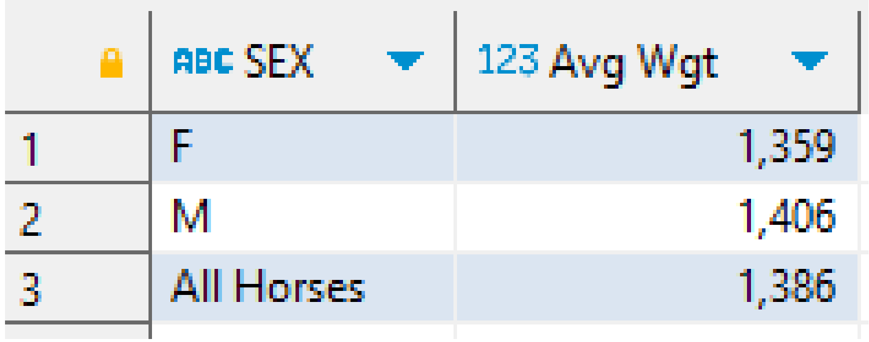


	123 GROUPING(SEX)	ABC SEX	123 Avg Wgt
1	0	F	1,359
2	0	M	1,406
3	1	[NULL]	1,386

Using DECODE() function to add a label

Observe the use of the DECODE function with the GROUPING function to add a label to an otherwise null value.

```
SELECT DECODE(GROUPING(sex),1,'All Horses', sex) AS Sex,
round(avg(weight),0) AS "Avg Wgt"
FROM HORSES
GROUP BY ROLLUP(sex);
```



	SEX	Avg Wgt
1	F	1,359
2	M	1,406
3	All Horses	1,386

Module 13.4

Extended GROUP BY clauses

- ROLLUP
- CUBE
- GROUPING
- GROUPING SETS

Good tools to have in your “tool belt”

- If you are working with databases, you will use SELECT, FROM, WHERE, JOIN, LIKE, GROUP BY, ORDER BY, aggregate functions, etc… daily

- The queries we’ve covered today might be SUPER helpful someday, but you’ll probably want to look up the syntax when that time comes (and the syntax may vary depending on the DBMS)

Progress Quiz Time!

- The Progress Quiz is available in Canvas
    - You MUST complete the quiz on Canvas by 5:00 on Friday – This in-class activity does not count for points!

    - Each week we will discuss the questions, so for those of you that are in class and keeping up with things, you’ll have an extra easy time with it!

- Go to http://kahoot.it and we’ll get started momentarily!

Break

What did we cover in this class?

- Basic database concepts
- Entity Relationship Diagrams
- Relational Data Modeling

A walk down memory lane

- SQL
- Normalization

Exam 2 content

Review: Data vs. Information

Data is raw/unformatted/unorganized

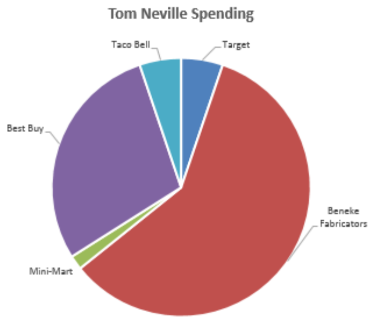
```
12012012,345844475,2295,2213,140223
12012012,345844475,1245,25100,115123
12012012,427658847,1154,885,57625
12052012,345844475,3011,754,114369
12062012,427658847,9584,10001,47624
12082012,427658847,2295,2523,45101
12122012,345844475,9584,12245,101217
12152012,345844475,1154,1300,99917
12192012,345844475,1154,907,113462
12192012,427658847,2224,1085,44016
12192012,427658847,1154,975,43041
12222012,427658847,2224,1085,41956
12231012,427658847,3030,122,41834
12262012,427658847,2295,1850,39984
12272012,427658847,1199,1925,38059
12272012,427658847,2224,1085,36974
12292012,427658847,9999,2000,34974
```

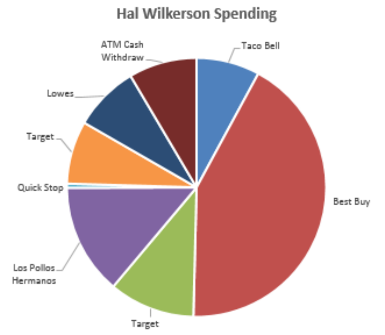
Data in context = Information

Date	Cust_ID	Vend_ID	Charge	Balance
12-01-2012	345-84-4475	2295	$22.13	$1,402.23
12-01-2012	345-84-4475	1245	$251.00	$1,151.23
12-01-2012	427-65-8847	1154	$8.85	$576.25
12-05-2012	345-84-4475	3011	$7.54	$1,143.69
12-06-2012	427-65-8847	9584	$100.01	$476.24
12-08-2012	427-65-8847	2295	$25.23	$451.01
12-12-2012	345-84-4475	9584	$122.45	$1,012.17
12-15-2012	345-84-4475	1154	$13.00	$999.17
12-19-2012	345-84-4475	1154	$9.07	$1,134.62
12-19-2012	427-65-8847	2224	$10.85	$440.16
12-19-2012	427-65-8847	1154	$9.75	$430.41
12-22-2012	427-65-8847	2224	$10.85	$419.56
12-23-1012	427-65-8847	3030	$1.22	$418.34
12-26-2012	427-65-8847	2295	$18.50	$399.84

Vend_ID	Vendor
1154	Taco Bell
1199	Lowes
1245	Beneke Fabricators
2224	Los Pollos Hermanos
2295	Target
3011	Mini-Mart
3030	Quick Stop
9584	Best Buy
9999	ATM Cash Withdraw

Cust_ID	Customer
345-84-4475	Tom Neville
427-65-8847	Hal Wilkerson

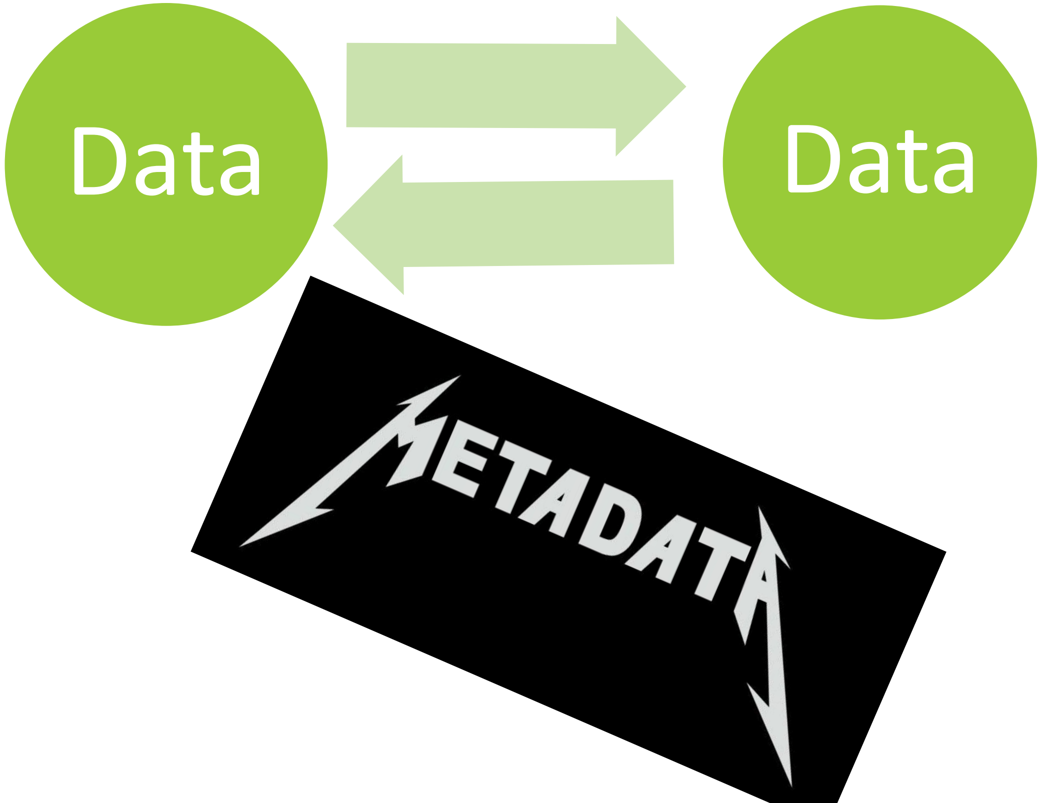




Knowledge = Information analyzed, visualized, etc. to help make decisions and predictions

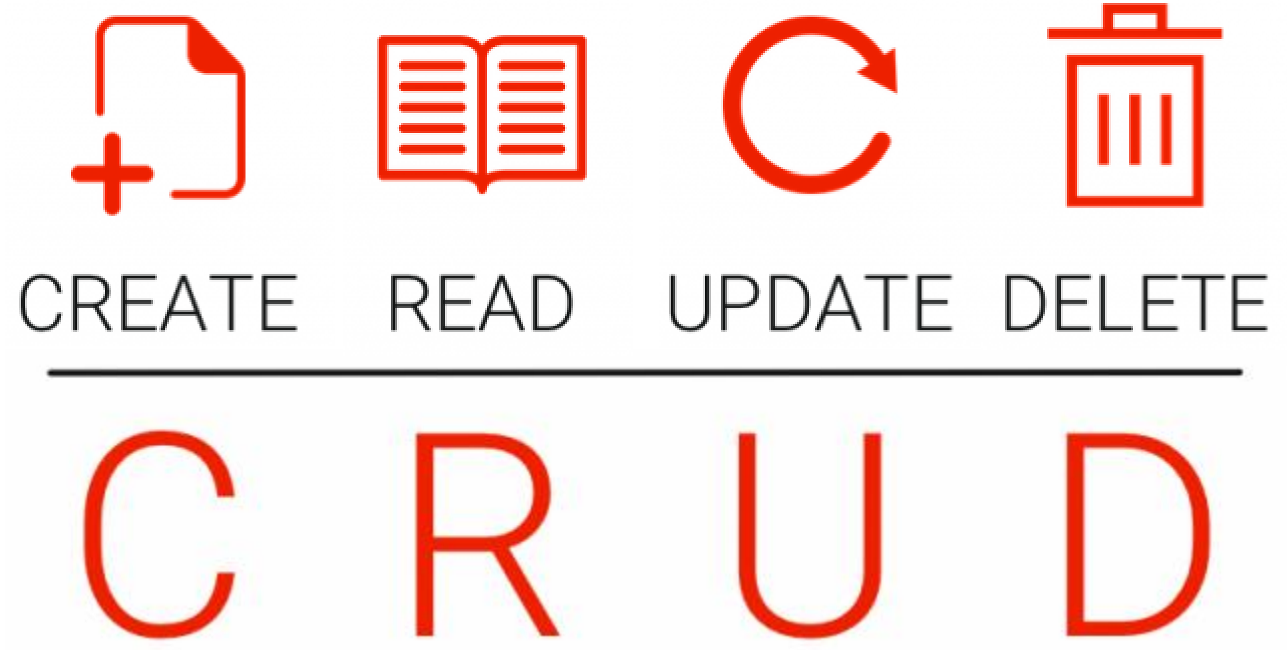
Metadata

Metadata is DATA that describes your DATA



Review: What are the four actions of data management?

- An unfortunate abbreviation:



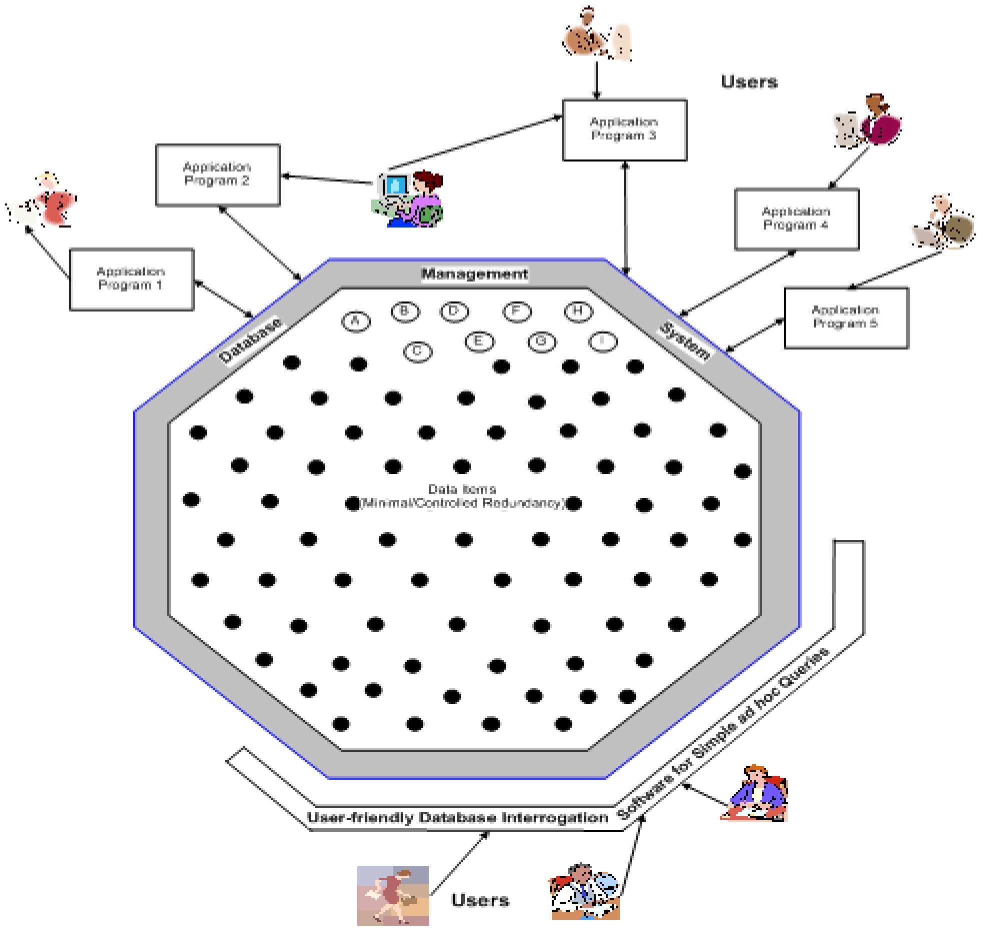
Review: What is a <u>D</u>ata<u>b</u>ase <u>M</u>anagement <u>S</u>ystem

- A DBMS Facilitates data access in a database without burdening a user with the details of how the data is physically organized

ORACLE®

Microsoft® SQL Server®

MySQL®



Review: What are the three levels of the ANSI/SPARC three schema architecture?

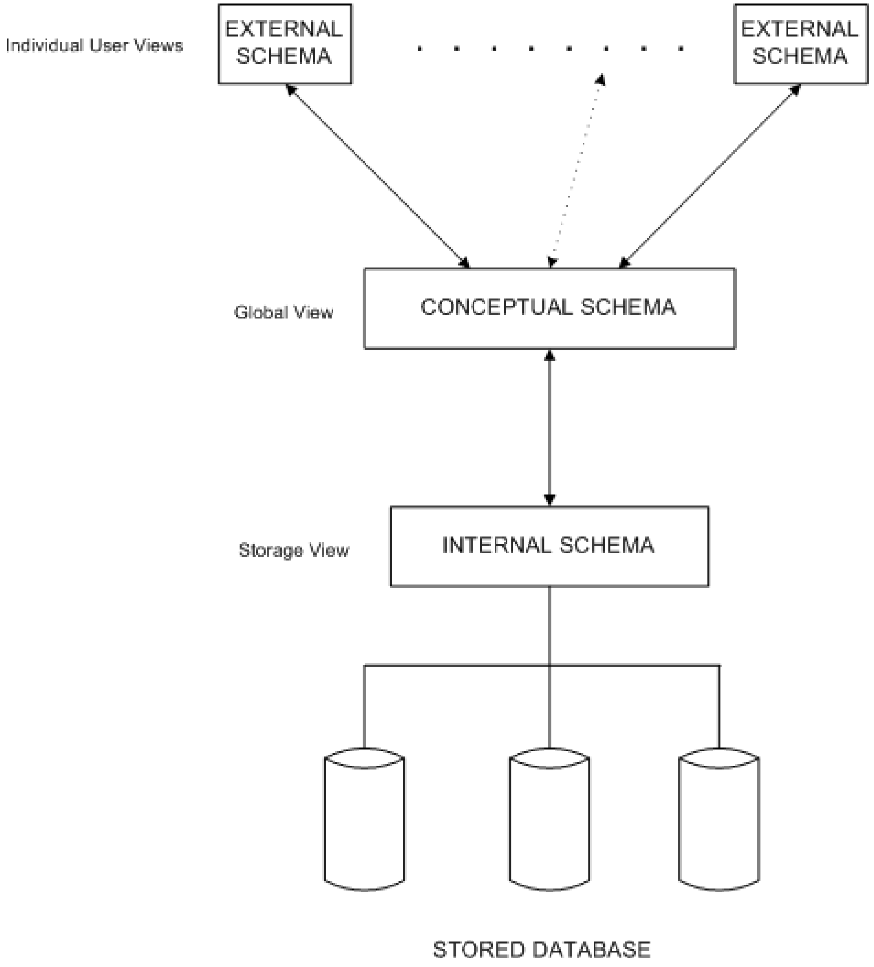


Figure 1.2 The ANSI/SPARC three-schema Architecture

Review: What is a model?

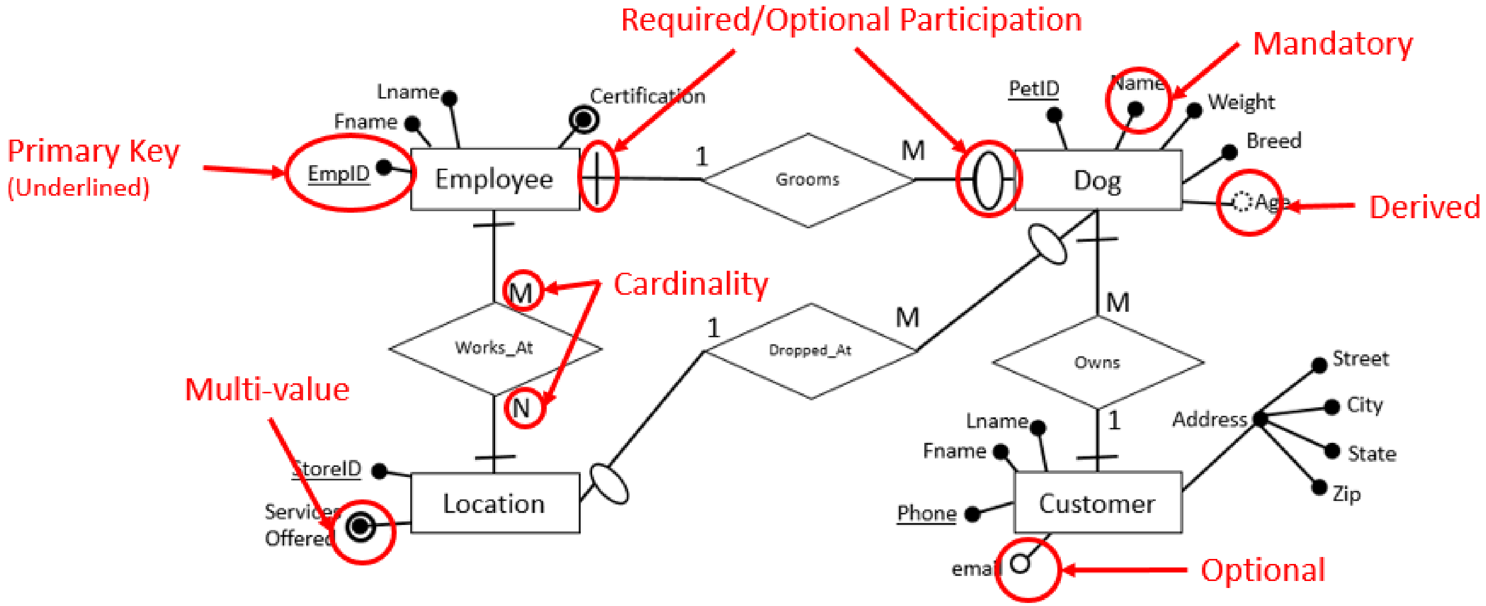
- Simplified expression of observed or unobservable reality used to perceive relationships in the outside world.

- All models are wrong, but some are useful
  - Box, George. E. P., and Draper, N. R., (1987), *Empirical Model Building and Response Surfaces*, John Wiley & Sons, New York, NY.

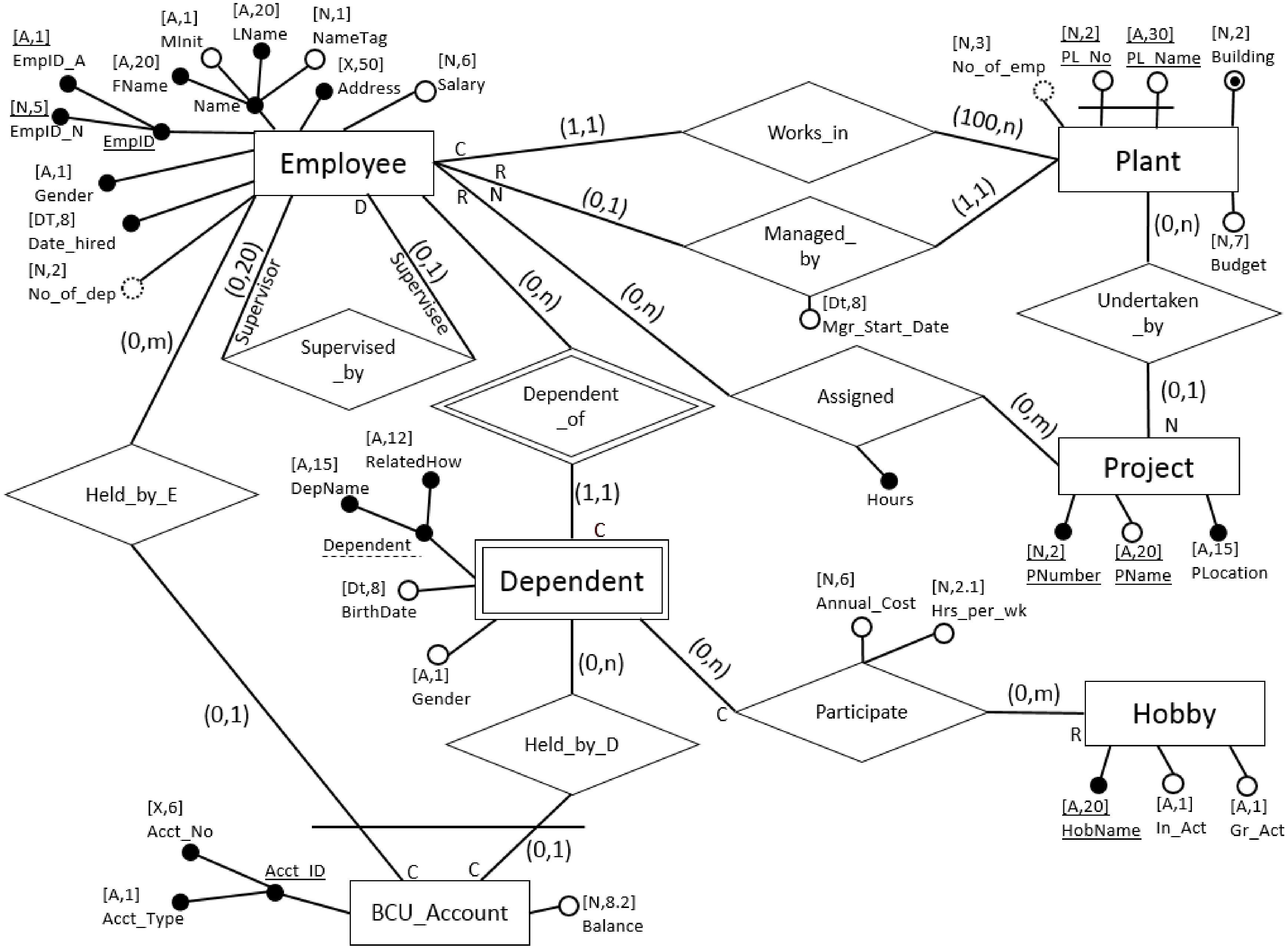
- If a model was perfectly correct, it would be the real thing!

Review: ER Model

- An ER diagram that portrays **entity types**, **attributes**, and **relationships** among entity types



- Semantic integrity constraints that reflect the **business rules** about data not captured in the ER diagram
  - The things circled in red
  - Sometimes cannot represent graphically, so we write them out


[A,1]
EmpID_A
[N,5]
EmpID_N
EmpID
[A,20]
FName
[A,1]
MInit
[A,20]
LName
[N,1]
NameTag
Name
[X,50]
Address
[N,6]
Salary
[A,1]
Gender
[DT,8]
Date_hired
[N,2]
No_of_dep
Employee
C
R
R
N
D
(1,1)
Works_in
(100,n)
[N,3]
No_of_emp
[N,2]
PL_No
[A,30]
PL_Name
[N,2]
Building
Plant
(0,n)
[N,7]
Budget
(0,1)
Managed_
by
(1,1)
[Dt,8]
Mgr_Start_Date
(0,20)
Supervisor
(0,1)
Supervisee
Supervised
_by
(0,n)
(0,n)
Undertaken
_by
(0,1)
N
(0,m)
Dependent
_of
Assigned
(0,m)
Hours
Project
[N,2]
PNumber
[A,20]
PName
[A,15]
PLocation
Held_by_E
(1,1)
C
[A,12]
RelatedHow
[A,15]
DepName
Dependent
[Dt,8]
BirthDate
Dependent
[A,1]
Gender
(0,n)
(0,n)
[N,6]
Annual_Cost
[N,2.1]
Hrs_per_wk
C
Participate
(0,m)
R
Hobby
[A,20]
HobName
[A,1]
In_Act
[A,1]
Gr_Act
(0,1)
Held_by_D
(0,1)
[X,6]
Acct_No
Acct_ID
[A,1]
Acct_Type
C
C
BCU_Account
[N,8.2]
Balance

Review: Cardinality – Parent-Child Relationships

- **One** coach (Parent) has **many** players
  - 1:n
  - **One** Dana Holgorsen coaches **many** players
  - Players have one coach

- **Many** children have **one** mother (Parent)
  - n:1
  - **Many** children are raised by **one** Kate Gosselin
  - Children have one mother

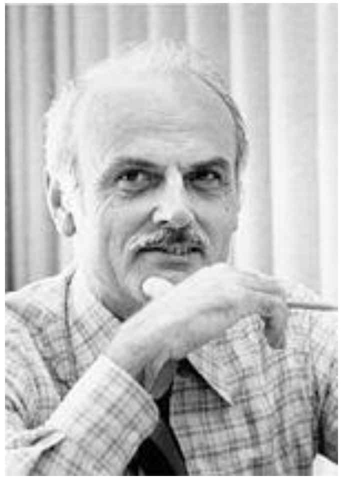
- **Many** shoppers visit **many** stores
  - m:n (1:n + n:1)
  - **One** shopper visits **many** stores +
    Many shoppers visit **one** store

- **One** department has **one** department chair
  - 1:1
  - **One** faculty (Dr. Johnson) is chair of **one** department (DISC)
  - A faculty member cannot chair two departments, and a department cannot have two faculty members as chair!

UNIVERSITY of
HOUSTON
C. T. BAUER COLLEGE of BUSINESS
Department of Decision & Information Sciences
DISC Department

Review: The Relational Data Model

- Edgar F. Codd in 1970 used the concept of mathematical relations to define the relational data model

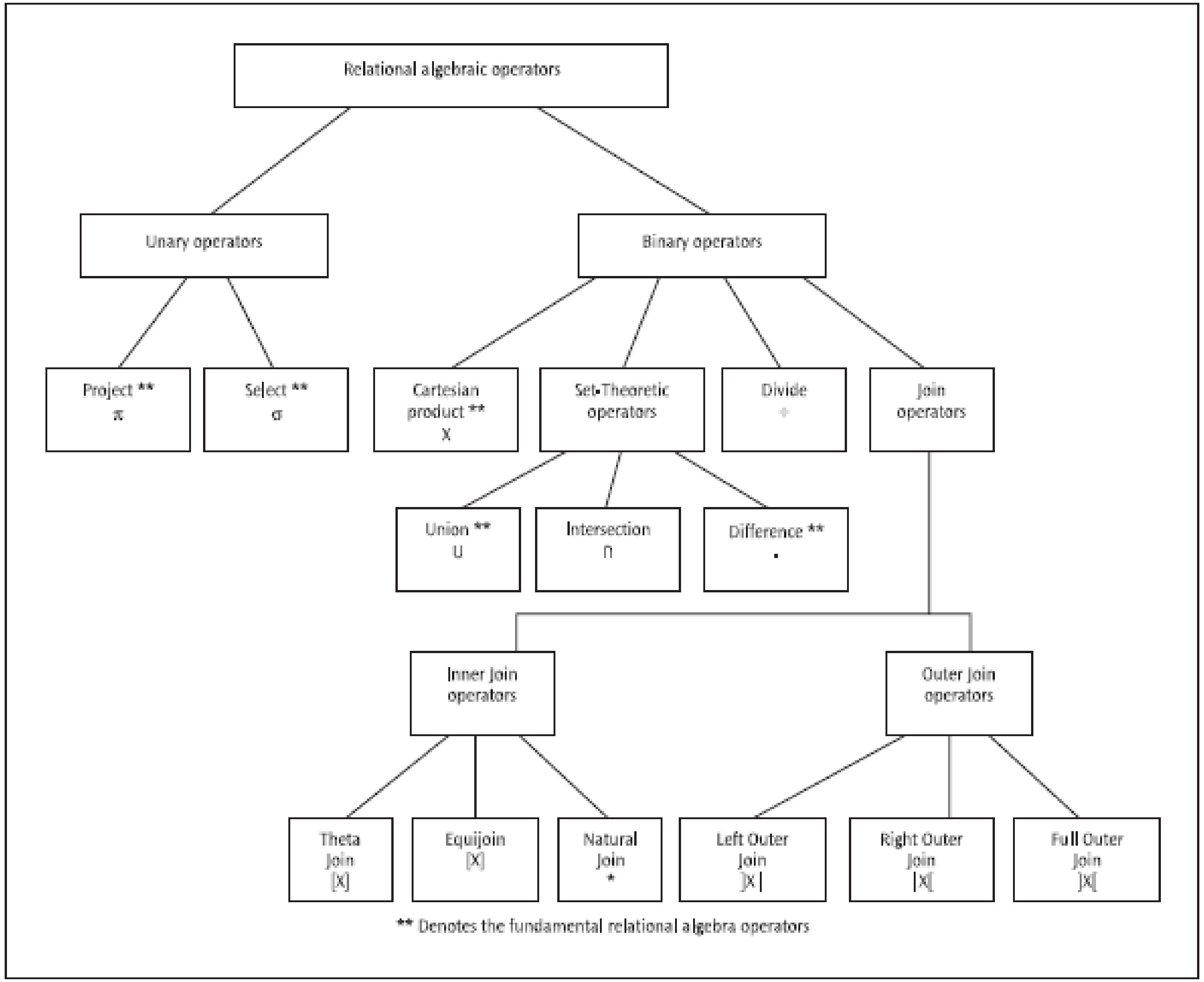


- Database = collection of relations
- Relation = two-dimensional table
- Tuple = Row of related data values in the table
- Attribute = Column in the table
- Domain = set of possible atomic values of an attribute

Review: Keys

Super keys	Uniquely identifies tuples
Candidate keys	Unique + irreducible
Key attributes	Proper subset of a Candidate Key
Non-key attributes	Not a subset of a CK
Primary key	Candidate Key with Entity Integrity Constraint (not NULL)
Alternate keys	A CK not selected as PK

Review: Relational algebra


Relational algebraic operators

Unary operators

Binary operators

Project ** π

Select ** σ

Cartesian product ** X

Set-Theoretic operators

Divide ÷

Join operators

Union ** ∪

Intersection ∩

Difference ** .

Inner Join operators

Outer Join operators

Theta Join |X|

Equijoin |X|

Natural Join *

Left Outer Join]X|

Right Outer Join |X[

Full Outer Join]X[

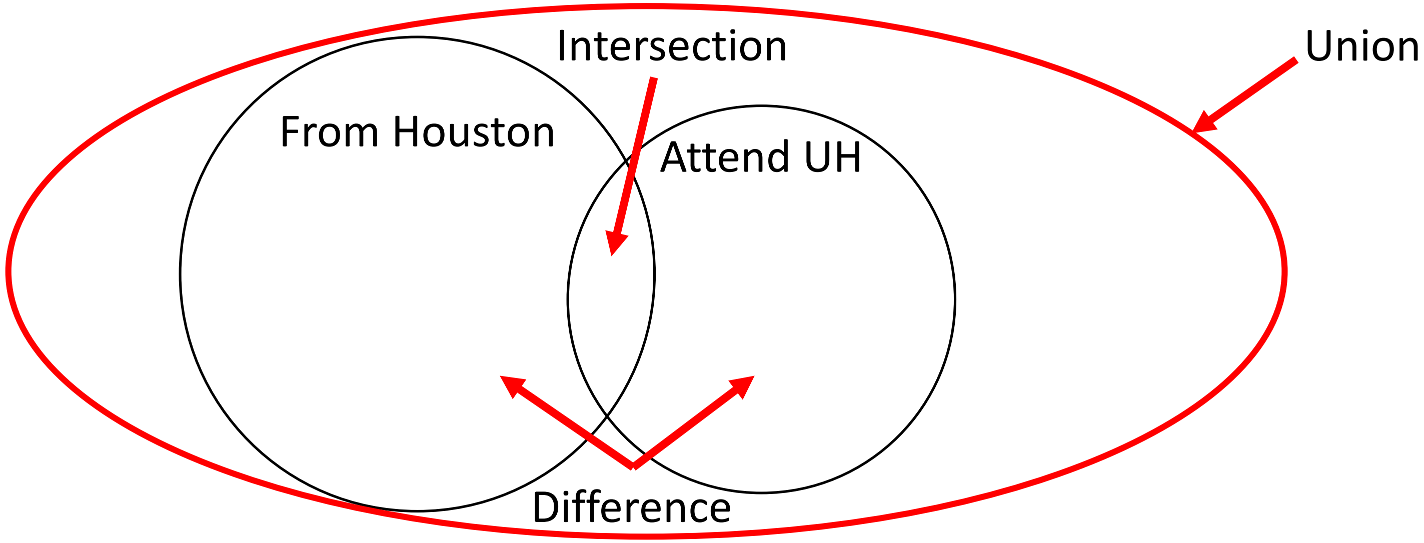
** Denotes the fundamental relational algebra operators

Figure 11.2 Classification of relational algebra operators

Review: Set Theory Operators

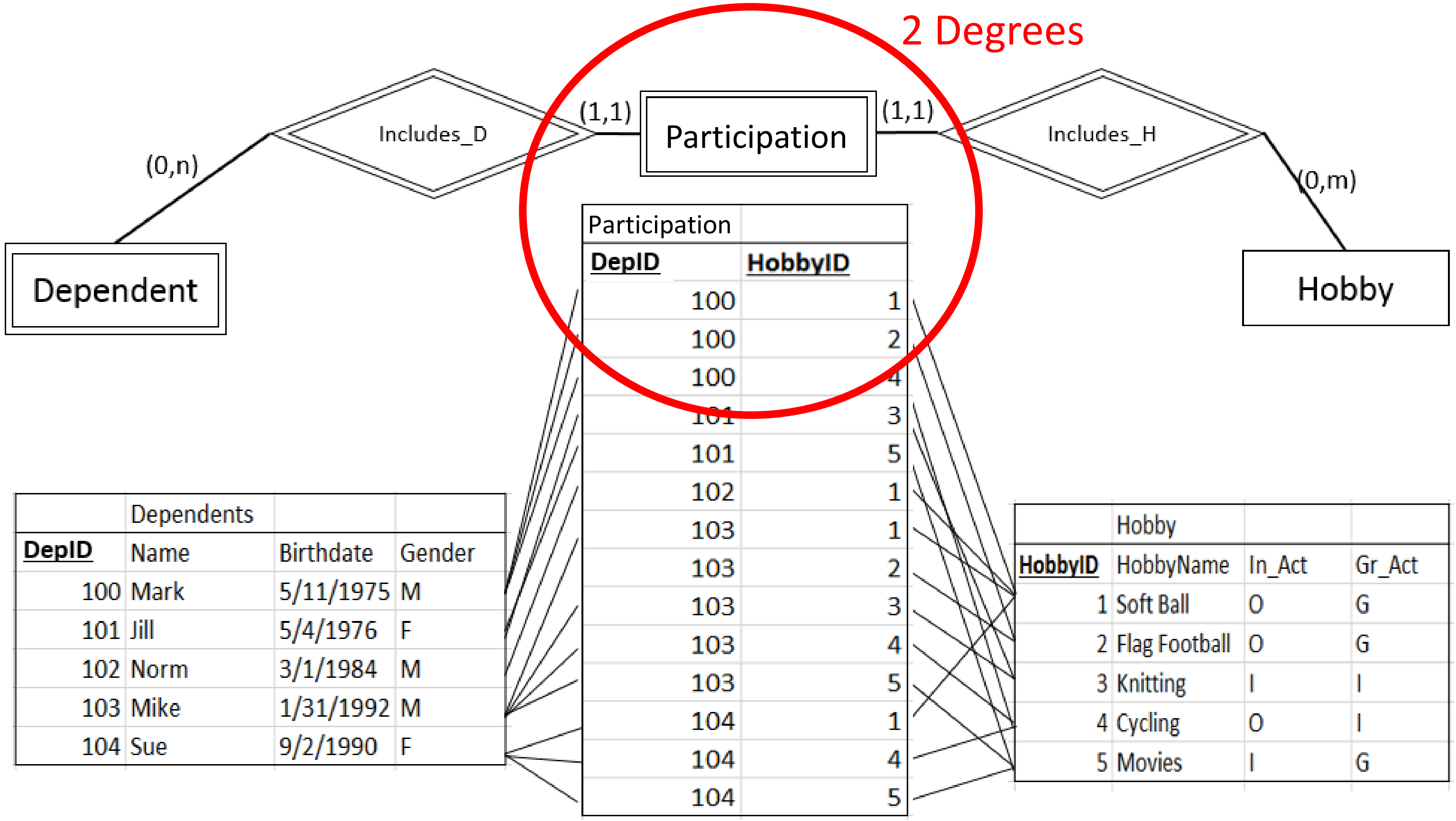
- Some people are from Houston
- Some people attend UH
- Union: People that are from Houston OR go to UH
- Intersection: People that are from Houston AND go to UH
- Difference: People from Houston but do NOT go to UH

  People that go to UH but NOT from Houston



Review: Decomposed M:N relationship

- When we decompose the m:n relationship, how many degrees are in the participants relation?

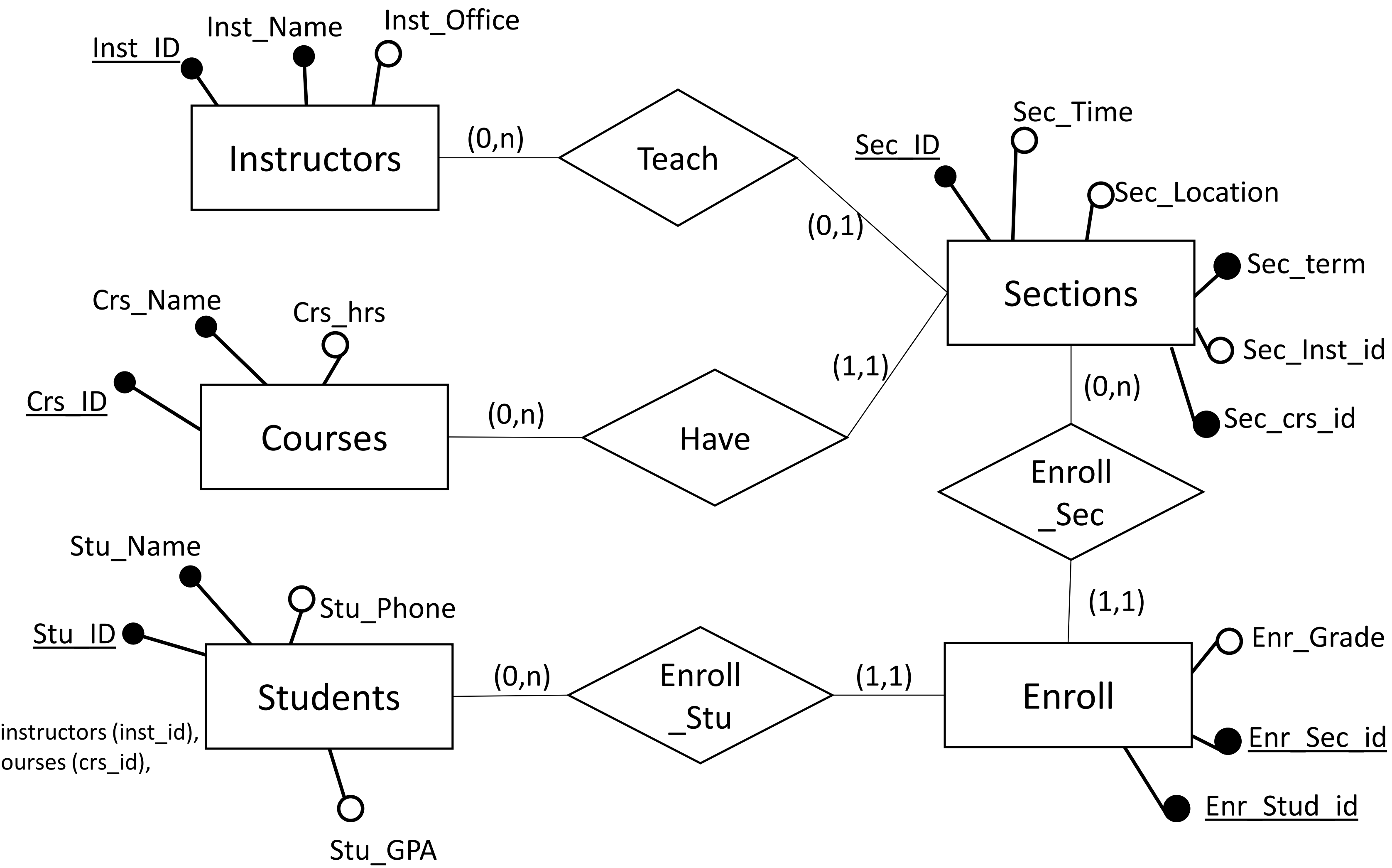


Participation	
DepID	**HobbyID**
100	1
100	2
100	4
101	3
101	5
102	1
103	1
103	2
103	3
103	4
103	5
104	1
104	4
104	5

	Dependents		
DepID	Name	Birthdate	Gender
100	Mark	5/11/1975	M
101	Jill	5/4/1976	F
102	Norm	3/1/1984	M
103	Mike	1/31/1992	M
104	Sue	9/2/1990	F

	Hobby		
HobbyID	HobbyName	In_Act	Gr_Act
1	Soft Ball	O	G
2	Flag Football	O	G
3	Knitting	I	I
4	Cycling	O	I
5	Movies	I	G

ERD and DDL



```
CREATE TABLE instructors (
 Inst_id        char(15) CONSTRAINT pk_inst PRIMARY KEY,
 Inst_name      varchar(50) CONSTRAINT nn_Instname NOT NULL,
 Inst_office    varchar(8)
);
```

```
CREATE TABLE courses(
 Crs_id         char(8) CONSTRAINT pk_crs PRIMARY KEY,
 Crs_name       varchar(50) CONSTRAINT nn_crs NOT NULL,
 Crs_hours      numeric(1)
);
```

```
CREATE TABLE sections (
 Sec_id          char(5) CONSTRAINT pk_sec PRIMARY KEY,
 Sec_time        datetime,
 Sec_location    nvarchar(10),
 Sec_term        char(8) CONSTRAINT nn_secterm NOT NULL,
 Sec_inst_id     char(15) CONSTRAINT fk_inst FOREIGN KEY REFERENCES instructors (inst_id),
 Sec_crs_id     char(8) CONSTRAINT fk_cou FOREIGN KEY REFERENCES  courses (crs_id),
 CONSTRAINT nn_sec_crs NOT NULL
);
```

```
CREATE TABLE students (
 Stu_id         char(15) CONSTRAINT pk_stu PRIMARY KEY,
 Stu_name       nvarchar(50) CONSTRAINT nn_stuname NOT NULL,
 Stu_phone      char(12),
 Stu_GPA        numeric(1,3),
 CONSTRAINT chk_gpa CHECK (Stu_GPA <= 4.0)
);
```

```
CREATE TABLE enrollment(
Enr_stu_id     char(15),
Enr_sec_id     char(8),
Enr_Grade       char(2),
CONSTRAINT fk_stu FOREIGN KEY Enr_stu_id REFERENCES students (stu_id),
CONSTRAINT fk_sec FOREIGN KEY Enr_sec_id REFERENCES sections (sec_id),
CONSTRAINT pk_enr PRIMARY KEY (Enr_stu_id, Enr_sec_id),
);
```

What have we covered since the first exam?

- SQL
- Normalization

Basic SQL – You will write queries on the exam

- SELECT / FROM / WHERE
- LIKE
- ORDER BY
- GROUP BY
- Aggregate Queries
- HAVING
- String manipulation (SUBSTR, INSTR, LENGTH, concatenation, etc.)

- Cartesian Product
- INNER JOIN
- LEFT, RIGHT, and FULL OUTER JOIN
- Set operations
- Subqueries

Advanced SQL – You should recognize on the exam

- DECODE
- CASE
- Hierarchical Queries
- GROUP BY extensions
    - CUBE
    - ROLLUP
    - GROUPING SETS

A note for the exam

- For the “Advanced” SQL topics, you will **not** be required to produce the **SYNTAX** on the exam, however you should know what these operators do

- I may ask questions like “What does the rollup operator do?” or “What is Start With – Connect By used for?” but I will not ask you to **write** a SQL query using these operators

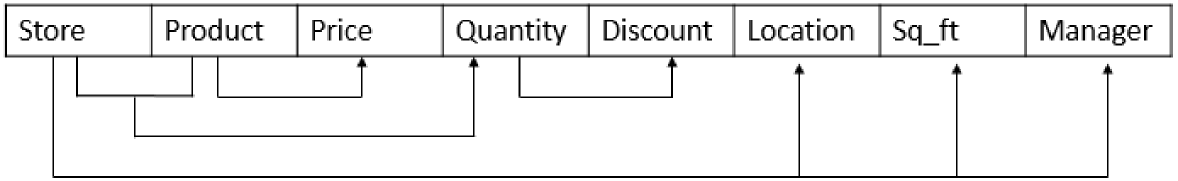
- All the topics on the “Basic SQL” slide are fair game for asking you to write SQL queries using any/all of these functions/topics.

Normalization

- Functional Dependencies / Dependency Diagrams
- Armstrong’s Axioms
- Identifying candidate keys (Synthesis and Decomposition)
- Identifying 1NF, 2NF, 3NF, and BCNF
- Normalizing to 3NF

Review: Functional Dependencies

- Functional dependencies (FD) specify a relationship between attributes in a relation schema
  - May be semantically obvious or Inferred



Store | Product | Price | Quantity | Discount | Location | Sq_ft | Manager

- Attribute **A** in a relation schema **R** *functionally determines* another attribute **B** in **R** if for a given value of **A** there is a single, specific value of **B**

- Expressed as A → B
  - A is called the **determinant**, B is called the **dependent**

- Undesirable function dependencies are the 'seeds' of data redundancy leading to modification anomalies.

Review: Armstrong’s Axioms

- Three primary Axioms
  - Reflexivity
    - If Y is a subset of X, then X → Y (a trivial dependency)
  - Augmentation
    - If X → Y, then {X,Z} → Y and {X,Z}→{Y,Z}
  - Transitivity
    - If X → Y and Y → Z, then X → Z

- Four inference rules
  - Union
    - If X → Y and X → Z, then X → {Y,Z}
  - Decomposition
    - If X → {Y,Z}, then X → Y and X → Z
  - Composition
    - If A → B and C → D, then {A,C} → {B,D}
  - Pseudotransitivity
    - If X → Y and {Y,W} → Z, then {X,W} → Z

Review: Normal Forms

- Normal Forms
  - 1NF
    - No multi value attributes
    - Meet the basic requirements to be a relation
  - 2NF
    - 1NF + No partial dependencies
  - 3NF
    - 2NF + No transitive dependencies
  - BCNF
    - 3NF + Every determinant must be a superkey
    - Removes the possibility of non-superkeys determining key attributes (which is OK in 3NF)

Review: First Normal Form (1NF)

- First normal form (1NF) imposes conditions so a relation does not contain records with a variable number of fields.
  - This is accomplished by prohibiting multi-valued and composite attributes in a relation schema.

- Such a constraint, in effect, prevents relations from containing other relations.

Second Normal Form (2NF)

Note: There is a confusing definition in the book!
On page 398 (and other places) the definition of 2NF says "non-prime" when it should say "non-key"!

- A relation schema R is in 2NF if every non-key attribute in R is fully functionally dependent on the primary key of R.

- This means a non-key attribute is not functionally dependent on a proper subset of the primary key of R.
  - Partial dependency

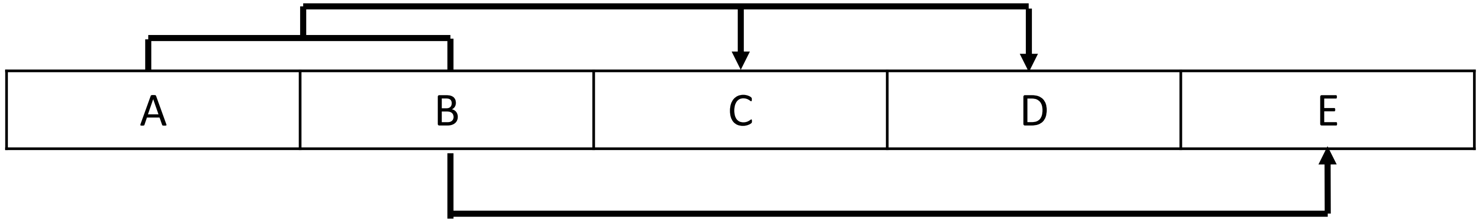
- The Second Normal Form (2NF) is based on a concept known as full functional dependency.

- A functional dependency of the form Z → A is a 'full functional dependency' if and only if no proper subset of Z functionally determines A.

- In other words, if Z → A and X → A, and X is a proper subset of Z, then Z does not fully functionally determine A, i.e., Z → A is not a full functional dependency; it is a partial dependency.

Review: Violation of 2NF

- {A,B} is the primary key
- {AB} → {CD} is fine
  - C and D are FULLY FUNCTIONALLY DEPENDENT on the PK
- B → E is a partial dependency
  - A non-prime attribute (E) is being determined by PART of the primary key



- The FD of B→E should be moved to a new relation
  - The determinant (B) will be left in the source relation

Third Normal Form – 3NF

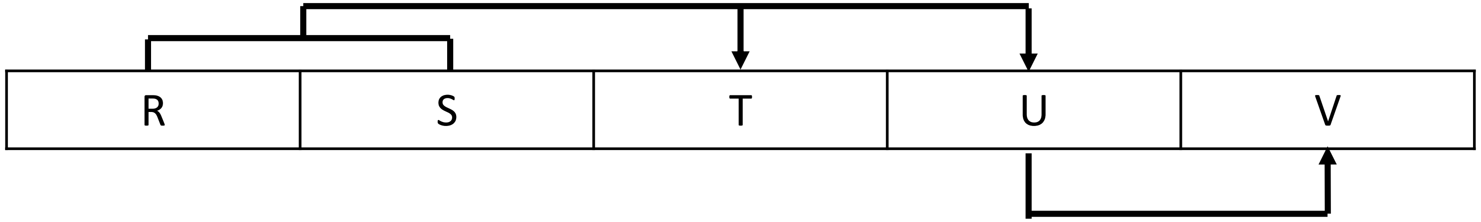
Note: There is a confusing definition in the book!
On page 402 (and other places) the definition of 3NF says "non-prime" when it should say "non-key"!

- A relation schema R is in 3NF if it is in 2NF and no non-key attribute is functionally dependent on another non-key attribute in R.
  - i.e., there are no transitive dependencies

- The Third Normal Form (3NF) is based on the concept of transitive dependency.
  - Given a relation schema R (X, A, B) where
    - X, A, and B are pair-wise disjoint atomic or composite attributes,
    - X is the primary key of R, and
    - A and B are non-key attributes
    - If A → B (or B → A) in R, then B (or A) is said to be 'transitively dependent' on X, the primary key of R.

Review: Violation of 3NF

- {R,S} is the primary key
- {RS} → {TU} is fine
  - T and U are FULLY FUNCTIONALLY DEPENDENT on the PK
- U → V is a transitive dependency
  - A non-key attribute (V) is being determined by another non-key attribute (U)



- The FD of U→V should be moved to a new relation
  - The determinant (U) will be left in the source relation

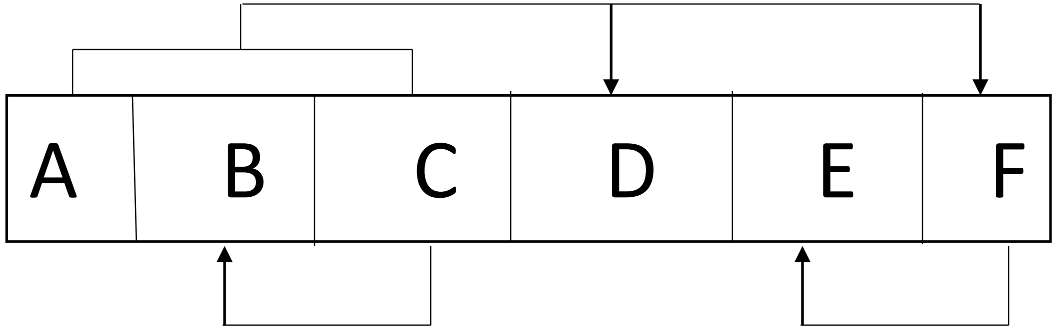
Normalization Problem

R1(A, B, C, D, E, F)

FD1: C→ B

FD2: F→E

FD3: {A,C} → {D,F}



CK is {A,C}

FD1 is a partial dependency

FD2 is a transitive dependency

3NF resolution:

R1(A, C, D, F)

FD3: {A,C} → {D,F}

R2(C, B)

FD1: C→ B

R3(E,F)

FD2: F→E

Now in R1, C is a foreign key that refers to C in R2, and F is a foreign key that refers to F in R3

For the exam

- Since we’ve covered a lot of stuff this semester, let me give some focus:

- Very important for the exam:
  - All the “basic” SQL we covered after spring break
  - All FD and normalization content including BCNF
  - SQL commands from today (single row character functions)
  - Of course understanding all the content from the first half of class is important to doing this stuff correctly, but I won’t ask about it directly

- The hierarchical queries, group by extensions, etc. are (in my experience) a little less common, and thus I am less likely to focus on these for the exam.
  - You should still review them and **there may be questions about one or more of them**, but they will not be a major focus of the exam.

Questions?

- If you have specific questions ask now… I am happy to pull up slides from previous lectures and discuss/review “anything”

What’s next in the world of data management?

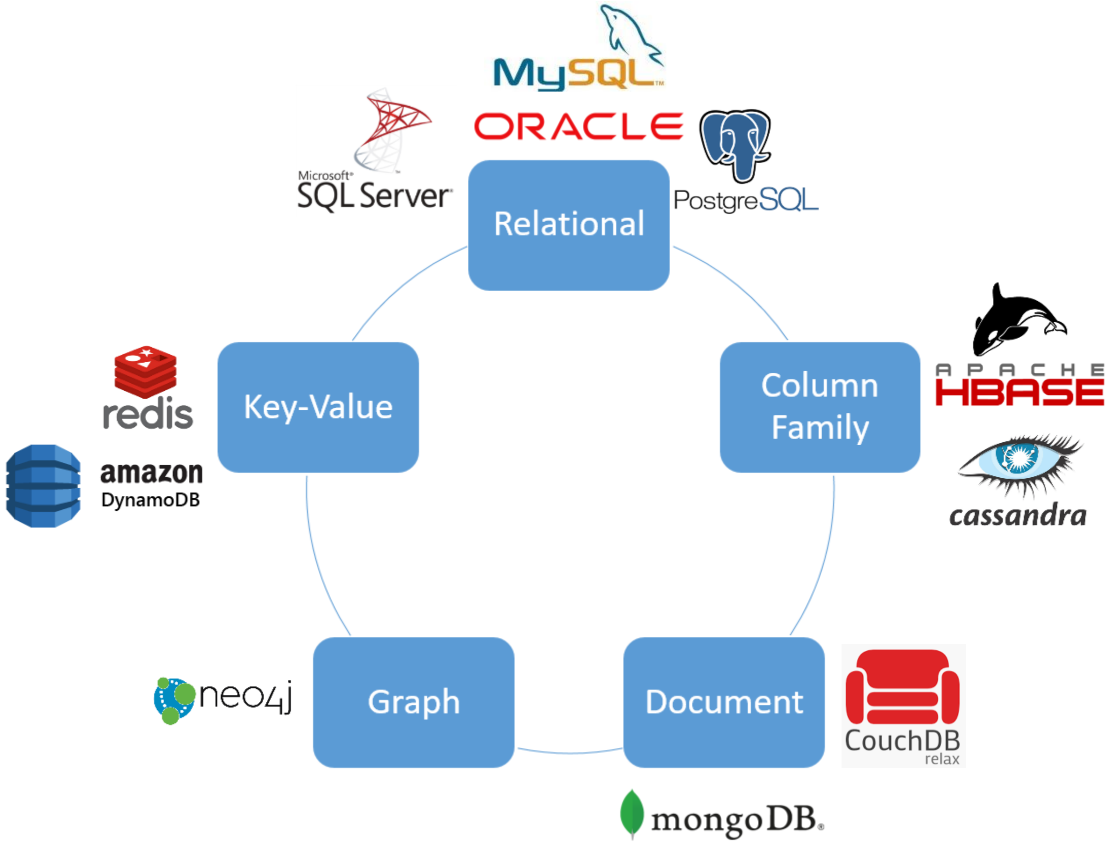
- Relational database have been around since 1970, and extremely widely used since the mid 1980’s

- Relational databases are still very important and widely used
  - …and IMO will continue to be for the foreseeable future

- HOWEVER, with the advent of “Web Scale” data, things are changing!
  - No one could have predicted millions of users spread across the world accessing the same data – relational database weren’t built for this

Interested in what’s next? Check out BZAN 6356

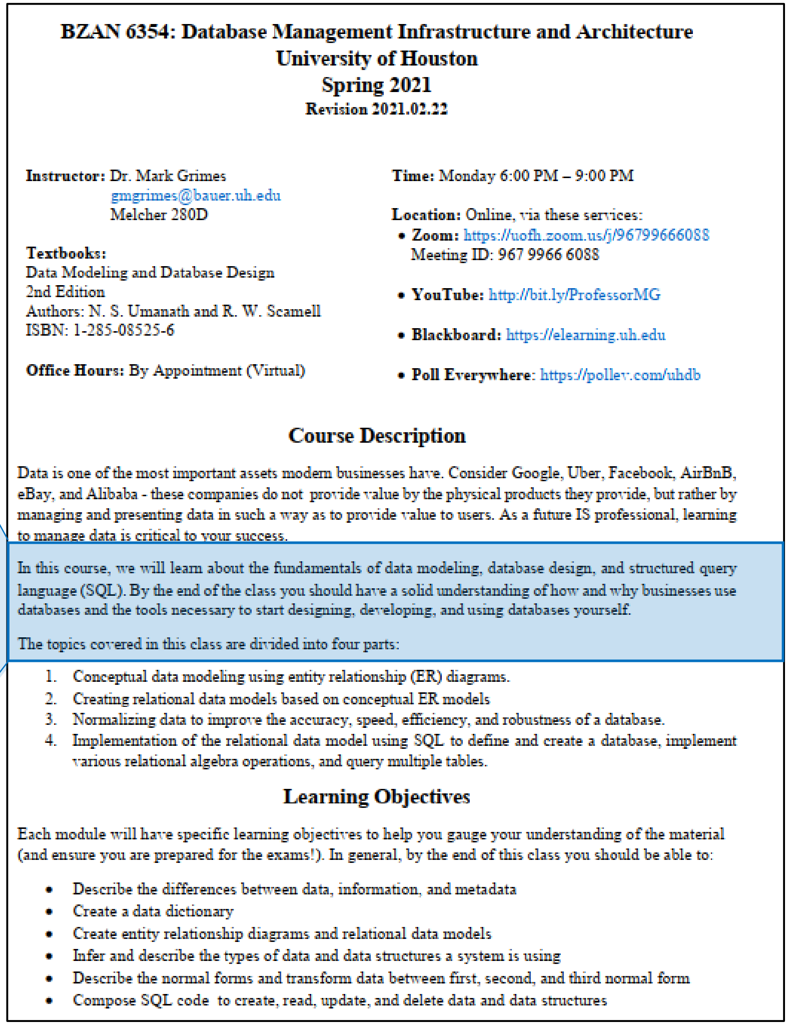
- Relational Databases
  - PostgreSQL
- Column Family Databases
  - HBase
- Document Databases
  - MongoDB
  - CouchDB
- Graph Databases
  - Neo4j
- Key-Value Databases
  - Redis
  - Amazon DynamoDB
- A variety of cloud-related topics



Did we accomplish what we set out to?

From page one of the syllabus:

In this course, we will learn about the fundamentals of data modeling, database design, and structured query language (SQL). By the end of the class you should have a solid understanding of how and why businesses use databases and the tools necessary to start designing, developing, and using databases yourself.



BZAN 6354: Database Management Infrastructure and Architecture
University of Houston
Spring 2021
Revision 2021.02.22

Instructor: Dr. Mark Grimes
gmgrimes@bauer.uh.edu
Melcher 280D

Textbooks:
Data Modeling and Database Design
2nd Edition
Authors: N. S. Umanath and R. W. Scamell
ISBN: 1-285-08525-6

Office Hours: By Appointment (Virtual)

Time: Monday 6:00 PM – 9:00 PM

Location: Online, via these services:

- **Zoom:** https://uofh.zoom.us/j/96799666088
  Meeting ID: 967 9966 6088
- **YouTube:** http://bit.ly/ProfessorMG
- **Blackboard:** https://elearning.uh.edu
- **Poll Everywhere**: https://pollev.com/uhdb

Course Description

Data is one of the most important assets modern businesses have. Consider Google, Uber, Facebook, AirBnB, eBay, and Alibaba - these companies do not provide value by the physical products they provide, but rather by managing and presenting data in such a way as to provide value to users. As a future IS professional, learning to manage data is critical to your success.

In this course, we will learn about the fundamentals of data modeling, database design, and structured query language (SQL). By the end of the class you should have a solid understanding of how and why businesses use databases and the tools necessary to start designing, developing, and using databases yourself.

The topics covered in this class are divided into four parts:

1. Conceptual data modeling using entity relationship (ER) diagrams.
2. Creating relational data models based on conceptual ER models
3. Normalizing data to improve the accuracy, speed, efficiency, and robustness of a database.
4. Implementation of the relational data model using SQL to define and create a database, implement various relational algebra operations, and query multiple tables.

Learning Objectives

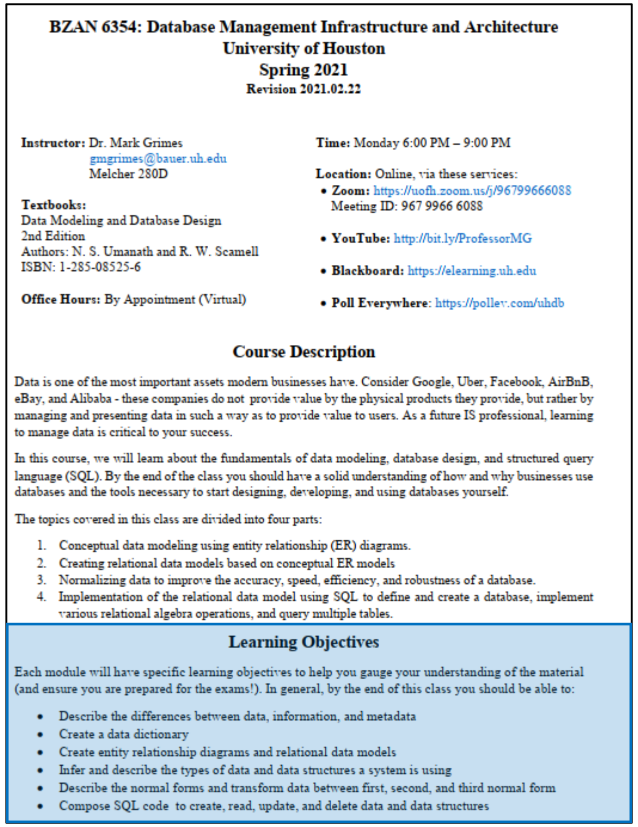
Each module will have specific learning objectives to help you gauge your understanding of the material (and ensure you are prepared for the exams!). In general, by the end of this class you should be able to:

- Describe the differences between data, information, and metadata
- Create a data dictionary
- Create entity relationship diagrams and relational data models
- Infer and describe the types of data and data structures a system is using
- Describe the normal forms and transform data between first, second, and third normal form
- Compose SQL code to create, read, update, and delete data and data structures

Hopefully we accomplished this!

Learning Objectives from the class

- Describe the differences between data, information, and metadata
  - First day of the course
- Create a data dictionary
  - Assignment 1
- Create entity relationship diagrams
  - First half of the course, Assignment 2 , Bearcat, etc…
- Create relational data models
  - First half of the course, Assignment 2 , Bearcat, etc…
- Infer and describe the types of data and data structures a system is using
  - Assignment 1, Assignment 2, Bearcat, various examples, etc…
- Describe the normal forms and transform data between 1NF, 2NF, and 3NF
  - Second half of the course, Assignment 4
- Compose SQL code to create, read, update, and delete data and data structures
  - Second half of the course, Assignment 3, SQL project



BZAN 6354: Database Management Infrastructure and Architecture
University of Houston
Spring 2021
Revision 2021.02.22

Instructor: Dr. Mark Grimes
gmgrimes@bauer.uh.edu
Melcher 280D

Textbooks:
Data Modeling and Database Design
2nd Edition
Authors: N. S. Umanath and R. W. Scamell
ISBN: 1-285-08525-6

Office Hours: By Appointment (Virtual)

Time: Monday 6:00 PM – 9:00 PM

Location: Online, via these services:
- **Zoom:** https://uofh.zoom.us/j/96799666088
  Meeting ID: 967 9966 6088
- **YouTube:** http://bit.ly/ProfessorMG
- **Blackboard:** https://elearning.uh.edu
- **Poll Everywhere**: https://pollev.com/uhdb

Course Description

Data is one of the most important assets modern businesses have. Consider Google, Uber, Facebook, AirBnB, eBay, and Alibaba - these companies do not provide value by the physical products they provide, but rather by managing and presenting data in such a way as to provide value to users. As a future IS professional, learning to manage data is critical to your success.

In this course, we will learn about the fundamentals of data modeling, database design, and structured query language (SQL). By the end of the class you should have a solid understanding of how and why businesses use databases and the tools necessary to start designing, developing, and using databases yourself.

The topics covered in this class are divided into four parts:

1. Conceptual data modeling using entity relationship (ER) diagrams.
2. Creating relational data models based on conceptual ER models
3. Normalizing data to improve the accuracy, speed, efficiency, and robustness of a database.
4. Implementation of the relational data model using SQL to define and create a database, implement various relational algebra operations, and query multiple tables.

Learning Objectives

Each module will have specific learning objectives to help you gauge your understanding of the material (and ensure you are prepared for the exams!). In general, by the end of this class you should be able to:

- Describe the differences between data, information, and metadata
- Create a data dictionary
- Create entity relationship diagrams and relational data models
- Infer and describe the types of data and data structures a system is using
- Describe the normal forms and transform data between first, second, and third normal form
- Compose SQL code to create, read, update, and delete data and data structures

Questions about the exam?

- Monday April 29 during class time in this room
  - In class on paper
  - 75 minutes
  - Closed Book / Individual Effort

- Will be approximately one third each of:
  - Multiple choice
  - Short answer / Matching / Fill in the Blank / etc.
  - Writing SQL and Normalization Problem(s)

<u>THERE IS NO FINAL EXAM DURING EXAM WEEK</u>

Go forth and do great things!

Hopefully you have found value in all this

Remember to do the course evaluation

(Link is in AccessUH)

BZAN 6354

Lecture 13

April 22, 2024

Dr. Mark Grimes, Ph.D.
gmgrimes@bauer.uh.edu

UNIVERSITY of
HOUSTON
C. T. BAUER COLLEGE of BUSINESS
Department of Decision & Information Sciences