L05: SQL: Advanced

CS3200 Database design (fa18 s2)

https://northeastern-datalab.github.io/cs3200/

Version 9/20/2018

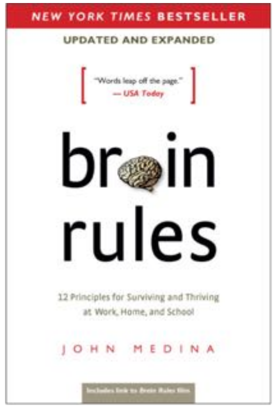
Announcements!

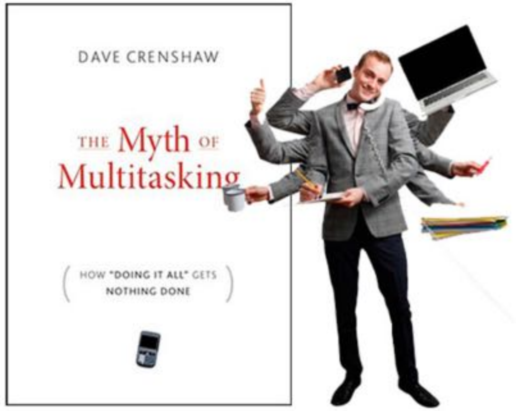
- HWs in this class are here for you to learn. Not for me to test you. Thus you may see topics for for which you have to read the textbook on your own.
  - Think about the rock-hammer-nail example from lecture 1
- Feel free to start working on your last HW (= exam 3 from last year)
- HW1 feedback: query execution took too long: next HWs smaller instances
- HW2 groups are assigned
- Student feedback: Speed: too fast?
- Class participation points for tips on increasing class participation (Surfer ...)
- Class participation: random calls

The "Surfer Analogy" for time management

Source: http://stwww.surfermag.com/files/2013/10/Yak_Charlie-970x646.jpg

Multitasking





"Myth #3: Multitasking when it comes to paying attention, is a myth… studies show that a person who is interrupted takes 50% longer to accomplish a task. Not only that, he or she makes up to 50% more errors" -- John Medina (Brain rules)

"…multitasking is a lie. You're asking me to switch attention, and that makes me less productive." -- Dave Crenshaw (The myth of multitasking)

"multitasking adversely affects how you learn. Even if you learn while multitasking, that learning is less flexible and more specialized, so you cannot retrieve the information as easily." --Russell Poldrack, UCLA Psychology Professor

"Our research offers neurological evidence that the brain cannot effectively do two things at once." -- Rene Marois, Dept. of Psychology, Vanderbilt

"The brain is a lot like a computer. You may have several screens open on your desktop, but you're able to think about only one at a time." -- William Stixrud, Neuropsychologist

If you do something else in class → I will pick on you:
You need to prove to me that you can multitask.

Source: Courtesy of Mike Smith

An example of SQL semantics

R

A
1
3

S

B	C
2	3
3	4
3	5

```sql
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

Cross Product

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

Apply Selections / Conditions

A	B	C
3	3	4
3	3	5

Apply Projection

Output

A
3
3

Note the semantics of a join

```
SELECT R.A
FROM   R, S
WHERE  R.A = S.B
```

1. Take **cross product**:

$$X = R \times S$$

Recall: Cross product (A X B) is the set of all unique tuples in A,B

Ex: {a,b,c} X {1,2}
= {(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)}

2. Apply **selections / conditions**:

$$Y = \{(r, s) \in X \mid r.A = r.B\}$$

= Filtering!

3. Apply **projections** to get final output:

$$Z = (y.A,)\ for\ y \in Y$$

= Returning only *some* attributes

We have seen that remembering this order is critical to understanding the output of certain queries

Note: we say “semantics” not “execution order”

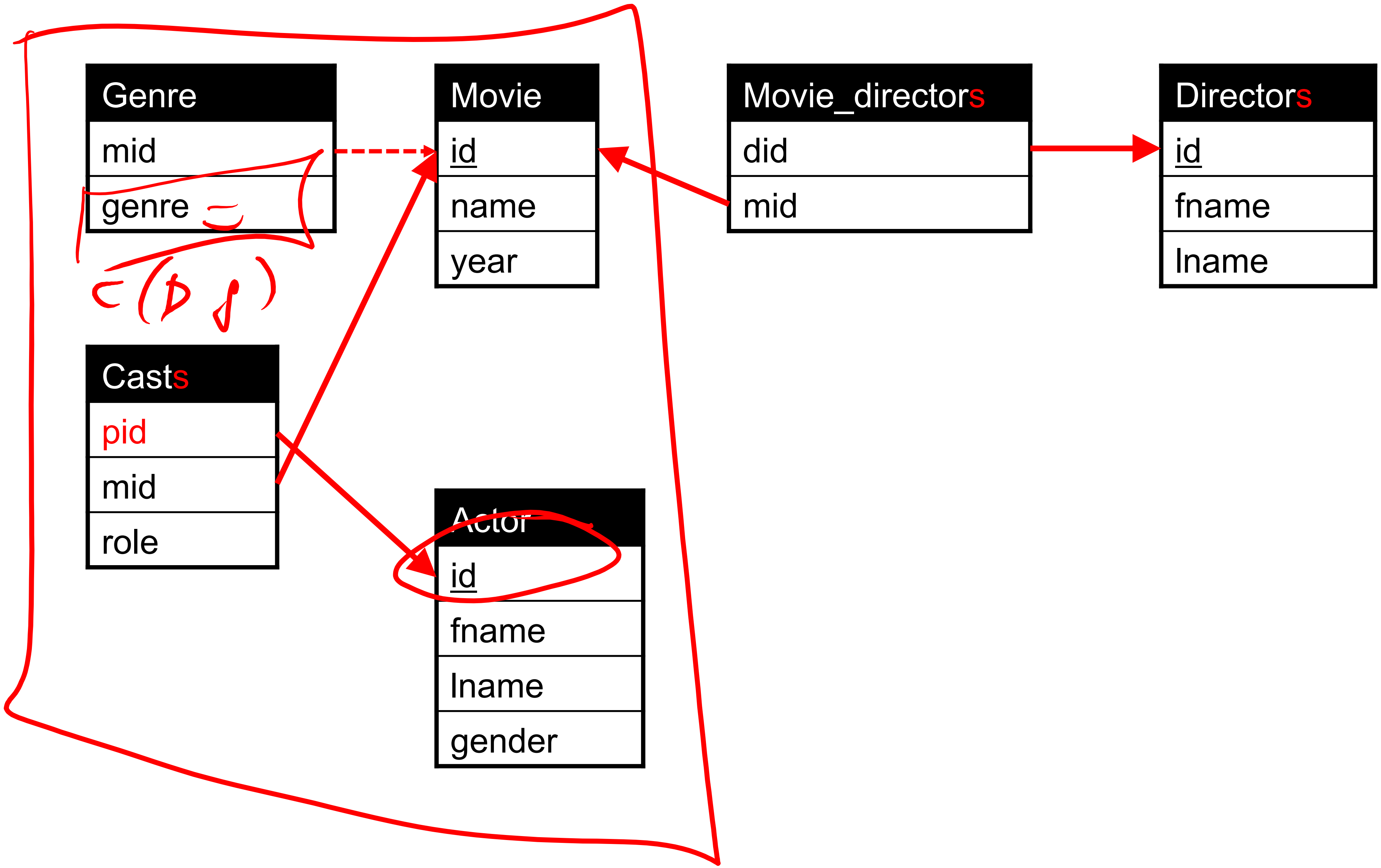
- The preceding slides show what a join means

- Not actually how the DBMS executes it under the covers

Data independence

- Logical data independence:
  - specify a set of attributes, not the logical navigation path to compute the connection among them
- Physical data independence:
  - specify a query, not the physical access paths to compute it

Big IMDB schema (Postgres)



3. Subqueries in WHERE (existential)

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

Using IN:

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid IN ( 1, 2 )
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

3. Subqueries in WHERE (existential)


315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make ***some*** *products with price < 25!*

Using IN:

"Set membership"

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid IN ( SELECT  P.cid
                   FROM    Product2 P
                   WHERE   P.price < 25)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

3. Subqueries in WHERE (existential)

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

EXISTS is true iff the subquery's result is not empty

Using EXISTS:

"Test for empty relations"

```
SELECT   DISTINCT C.cname
FROM     Company2 C
WHERE    EXISTS ( SELECT  *
                  FROM    Product2 P
                  WHERE   C.cid = P.cid
                    and   P.price < 25)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery

3. Subqueries in WHERE (existential)

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make <u>some</u> products with price < 25!

Using ANY (also some):

"Set comparison"

```
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   25 > ANY ( SELECT  price
                   FROM    Product2 P
                   WHERE   P.cid = C.cid)
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Correlated subquery

SQLlite does not support "ANY" ☹

3. Subqueries in WHERE (existential)

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Existential quantifiers ∃

Q: Find all companies that make some products with price < 25!

Now, let's unnest:

```
SELECT  DISTINCT C.cname
FROM    Company2 C, Product2 P
WHERE   C.cid = P.cid
   and  P.price < 25
```

cid	CName	City
1	GizmoWorks	Oslo
2	Canon	Osaka
3	Hitachi	Kyoto

PName	Price	cid
Gizmo	$19.99	1
Powergizmo	$29.99	1
SingleTouch	$14.99	2
MultiTouch	$203.99	3

Existential quantifiers are easy ! ☺

3. Subqueries in WHERE (universal)


315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Universal quantifiers ∀

Q: Find all companies that make only products with price < 25!

same as:

Q: Find all companies for which all products have price < 25!

Universal quantifiers are more complicated ! ☹
(Think about the companies that should not be returned)

3. Subqueries in WHERE (exist not -> universal)

Q: Find all companies that make only products with price < 25!

1. Find the other companies: i.e. they have some product $\geq$ 25!

```sql
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid IN (  SELECT  P.cid
                    FROM    Product2 P
                    WHERE   P.price >= 25)
```

2. Find all companies s.t. all their products have price < 25!

```sql
SELECT  DISTINCT C.cname
FROM    Company2 C
WHERE   C.cid NOT IN ( SELECT  P.cid
                       FROM    Product2 P
                       WHERE   P.price >= 25)
```

3. Subqueries in WHERE (exist not -> universal)


315

Product2 (pname, price, cid) Company2 (cid, cname, city)

Universal quantifiers ∀

Q: Find all companies that make <u>only</u> products with price < 25!

Using NOT EXISTS:

```
SELECT   DISTINCT C.cname
FROM     Company2 C
WHERE    NOT EXISTS ( SELECT   *
                      FROM     Product2 P
                      WHERE    C.cid = P.cid
                          and  P.price >= 25)
```

3. Subqueries in WHERE (exist not -> universal)

315

Product2 (pname, price, cid)
Company2 (cid, cname, city)

Universal quantifiers ∀

Q: Find all companies that make only products with price < 25!

Using ALL:

```
SELECT   DISTINCT C.cname
FROM     Company2 C
WHERE    25 > ALL (   SELECT   price
                      FROM     Product2 P
                      WHERE    P.cid = C.cid)
```

SQLlite does not support "ALL" ☹

Question for Database Fans & Friends

This topic goes beyond the course objectives;
only for those who are really interested
(computer science, research, grad school)

- How can we unnest the universal quantifier query ?

Queries that must be nested

This topic goes beyond the course objectives; only for those who are really interested (computer science, research, grad school)

- Definition: A query Q is monotone if:
  - Whenever we add tuples to one or more of the tables…
  - … the answer to the query cannot contain fewer tuples
- Fact: all unnested queries are monotone
  - Proof: using the "nested for loops" semantics
- Fact: Query with universal quantifier is not monotone
  - Add one tuple violating the condition. Then "all" returns fewer tuples
- Consequence: we cannot unnest a query with a universal quantifier

The drinkers-bars-beers example

331

Likes(drinker, beer)
Frequents(drinker, bar)
Serves(bar, beer)

Challenge: write these in SQL.
Solutions: http://queryviz.com/online/

Find drinkers that frequent some bar that serves some beer they like.

$$x: \quad \exists y.\ \exists z.\ \text{Frequents}(x, y) \wedge \text{Serves}(y,z) \wedge \text{Likes}(x,z)$$

Find drinkers that frequent only bars that serve some beer they like.

$$x: \quad \forall y.\ \text{Frequents}(x, y) \Rightarrow (\exists z.\ \text{Serves}(y,z) \wedge \text{Likes}(x,z))$$

Find drinkers that frequent some bar that serves only beers they like.

$$x: \quad \exists y.\ \text{Frequents}(x, y) \wedge \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Find drinkers that frequent only bars that serve only beer they like.

$$x: \quad \forall y.\ \text{Frequents}(x, y) \Rightarrow \forall z.(\text{Serves}(y,z) \Rightarrow \text{Likes}(x,z))$$

Null Values

3-valued logic example

- Three logicians walk into a bar. The bartender asks: "Do all of you want a drink?"
- The 1st logician says: "I don't know."
- The 2nd logician says: "I don't know."
- The 3rd logician says: "Yes!"

Nulls in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute if it can be NULL (nullable attribute) or not
- How does SQL cope with tables that have NULLs ?

Null Values

- In SQL there are three Boolean values:
  - FALSE, TRUE, UNKNOWN

- If x= NULL then
  - Arithmetic operations produce NULL. E.g: 4*(3-x)/7
  - Boolean conditions are also NULL. E.g: x='Joe'
  - aggregates ignore NULL values

- Logical reasoning:
  - FALSE = 0
  - TRUE = 1
  - UNKNOWN = 0.5

x AND y = min(x,y)

x OR y = max(x,y)

NOT x = (1 – x)

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
NULL	6.5	170

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL:
include only tuples that
yield TRUE

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL:
include only tuples that
yield TRUE

```
SELECT  *
FROM    Person
WHERE   age < 25  or age >= 25
```

Null Values: example

```
SELECT  *
FROM    Person
WHERE   (age < 25)
    and (height > 6 or weight > 190)
```

Person

Age	Height	Weight
20	NULL	200
~~NULL~~	~~6.5~~	~~170~~

Rule in SQL:
include only tuples that
yield TRUE

```
SELECT  *
FROM    Person
WHERE   age < 25  or age >= 25
```

Unexpected behavior

```
SELECT  *
FROM    Person
WHERE   age < 25  or age >= 25 or age IS NULL
```

Test NULL
explicitly

Null Values and Aggregates

T

gid	val
1	NULL
1	NULL
2	a
2	B
2	z
2	z
2	NULL
3	A
3	A
3	Z

```
SELECT gid,
       MAX(val) maxv,
       MIN(val) minv,
       COUNT(*) ctr,
       COUNT(val) ctv,
       COUNT(DISTINCT val) ctdv
FROM   T
GROUP BY gid
ORDER BY gid
```

?

Null Values and Aggregates


373

T

gid	val
1	NULL
1	NULL
2	a
2	B
2	z
2	z
2	NULL
3	A
3	A
3	Z

```
SELECT gid,
       MAX(val) maxv,
       MIN(val) minv,
       COUNT(*) ctr,
       COUNT(val) ctv,
       COUNT(DISTINCT val) ctdv
FROM   T
GROUP BY gid
ORDER BY gid
```

NULL is ignored by aggregate functions if you reference the column specifically. Exception: COUNT !

gid	maxv	minv	ctr	ctv	ctdv
1	NULL	NULL	2	0	0
2	z	B	5	4	3
3	Z	A	3	3	2

Null Values and Aggregates

T

gid	val
1	NULL
1	NULL
2	a
2	B
2	z
2	z
2	NULL
3	A
3	A
3	Z

```
SELECT val,
       COUNT(*) ctr
FROM   T
GROUP BY val
```

val	ctr
A	2
B	1
Z	1
a	1
z	2
NULL	3

NULL is included by "GROUP BY".
Relate sorting of NULL by "ORDER BY" is DBMS-specific

Side topic: sorting of strings

ASCII encoding

ASCII #	char
48	0
49	1
...	...
57	9
65	A
...	...
90	Z
97	a
...	...
122	z

```sql
SELECT 'A' < 'a' as eval
```

```sql
SELECT '1' < 'A' as eval
```

```sql
SELECT 'a' < 'ab' as eval
```

lexicographical order

eval
true

```sql
SELECT 'a' < 'A' as eval
```

eval
false

Inner Joins vs. Outer Joins

Illustration

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

An "inner join":

```
SELECT  *
FROM    English, French
WHERE   eid = fid
```

Same as:

```
SELECT  *
FROM    English JOIN French
ON      eid = fid
```

etext	eid	fid	ftext
One	1	1	Un
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz

"JOIN"
same as
"INNER JOIN"

Illustration

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit

"FULL JOIN"
same as
"FULL OUTER JOIN"

```
SELECT  *
FROM    English FULL JOIN French
ON      English.eid = French.fid
```

etext	eid	fid	ftext
One	1	1	Un
Two	2	NULL	NULL
Three	3	3	Trois
Four	4	4	Quatre
Five	5	5	Cinq
Six	6	6	Siz
NULL	NULL	7	Sept
NULL	NULL	8	Huit

~~SELECT *~~
~~FROM English JOIN French~~
~~ON eid = fid~~

SQLite does not support "FULL OUTER JOIN"s ☹ (but "LEFT JOIN")

Illustration

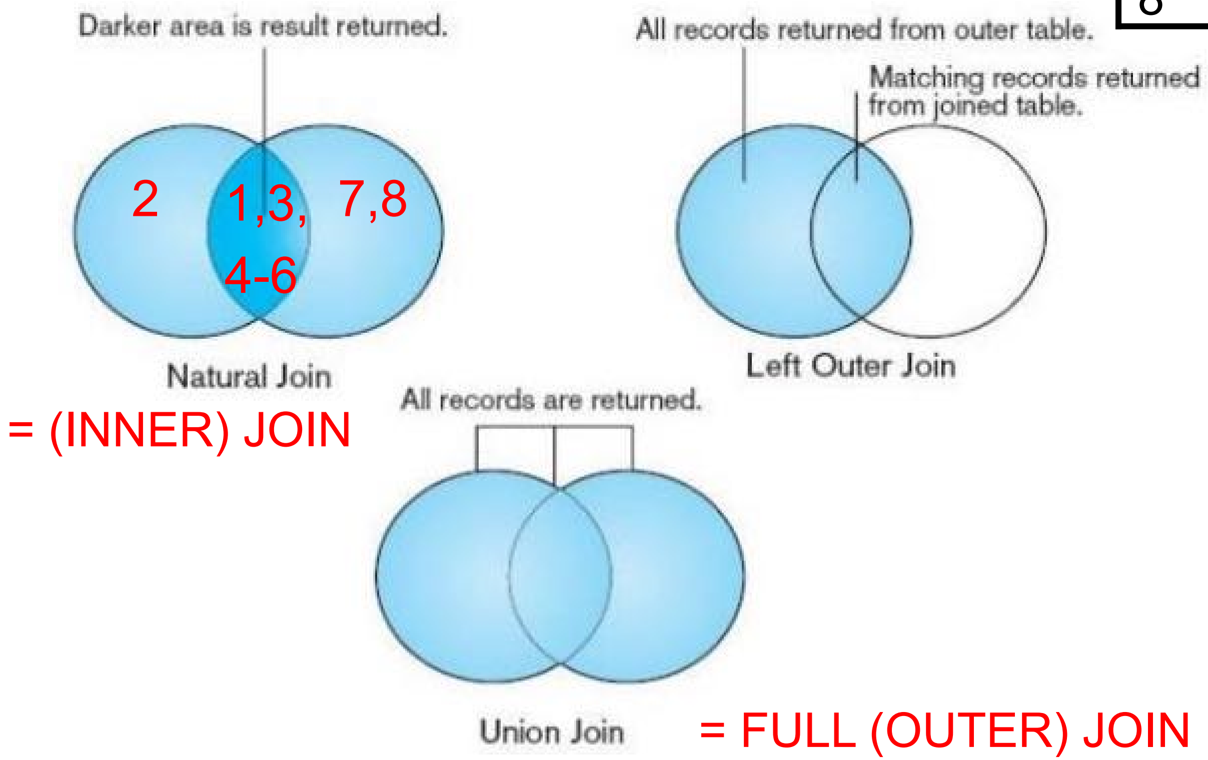
361

English

eText	eid
One	1
Two	2
Three	3
Four	4
Five	5
Six	6

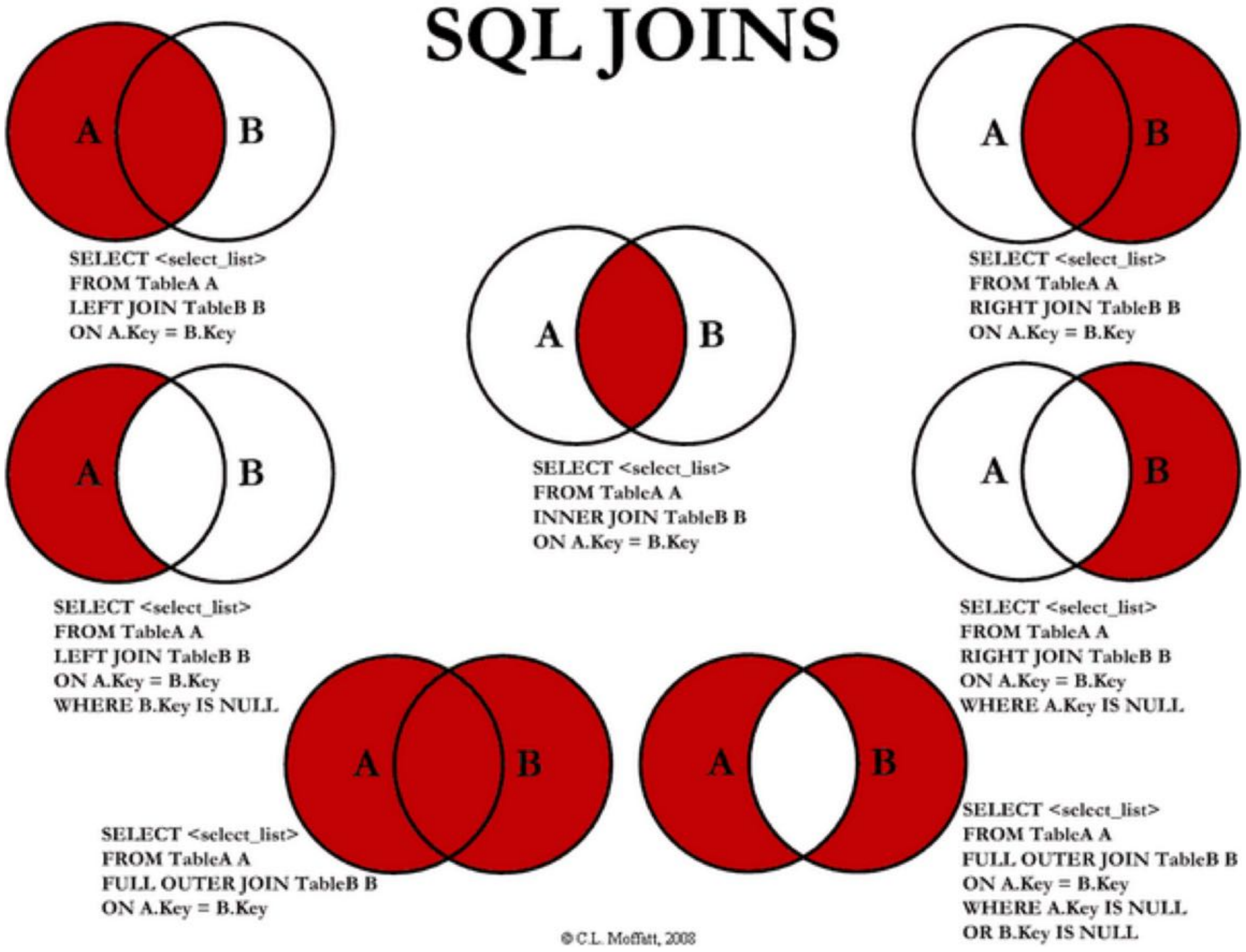
French

fid	fText
1	Un
3	Trois
4	Quatre
5	Cinq
6	Siz
7	Sept
8	Huit



Source: Fig. 7-2, Hoffer et al., Modern Database Management, 10ed ed, 2011.

Detailed Illustration with Examples (follow the link)



Check this web page for illustrating examples

Source: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins