UNIVERSITY *of* WASHINGTON

Week 4: SQL II: The Sequel

AGENDA

- Quiz Reminder
- Introduce Project
- SQL I Review & Examples
- More SQL
  - Subqueries
  - Relation Operators
  - Outer Joins
  - Grouping & Aggregation

SQL I REVIEW EXERCISES...

```
MOVIE (
    title:        string,
    year_made:    integer,
    length_mins:  integer,
    genre:        string,
    studio_name:  string,
    producer_num: integer
)
```

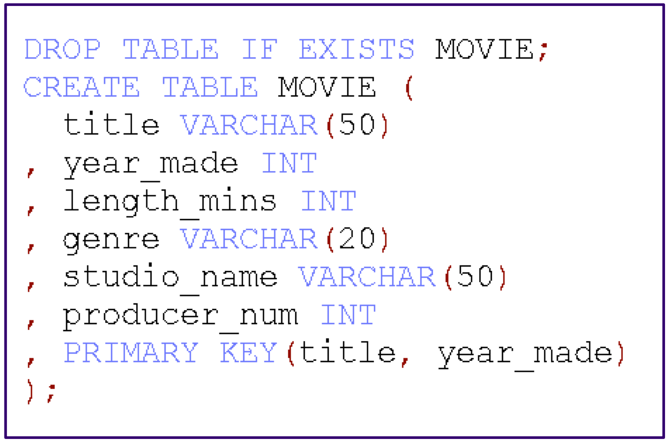
```
MOVIE_STAR (
    name:      string,
    address:   string,
    gender:    char,
    birthdate: date
)
```

```
STARS_IN (
    title:     string,
    year_made: integer,
    star_name: string
)
```

```
MOVIE_EXECUTIVE (
    full_name: string,
    address:   string,
    cert_num:  integer,
    net_worth: integer
)
```

```
STUDIO (
    legal_name:     string,
    address:        string,
    prez_cert_num:  integer
)
```

SQL I REVIEW EXERCISES…



```sql
DROP TABLE IF EXISTS MOVIE;
CREATE TABLE MOVIE (
  title VARCHAR(50)
, year_made INT
, length_mins INT
, genre VARCHAR(20)
, studio_name VARCHAR(50)
, producer_num INT
, PRIMARY KEY(title, year_made)
);
```

```
MOVIE_STAR (
    name:       string,
    address:    string,
    gender:     char,
    birthdate:  date
)
```

```
STARS_IN (
    title:      string,
    year_made:  integer,
    star_name:  string
)
```

```
MOVIE_EXECUTIVE (
    full_name:  string,
    address:    string,
    cert_num:   integer,
    net_worth:  integer
)
```

```
STUDIO (
    legal_name:     string,
    address:        string,
    prez_cert_num:  integer
)
```

SQL I REVIEW EXERCISES…

```
DROP TABLE IF EXISTS MOVIE;
CREATE TABLE MOVIE (
  title VARCHAR(50)
, year_made INT
, length_mins INT
, genre VARCHAR(20)
, studio_name VARCHAR(50)
, producer_num INT
, PRIMARY KEY(title, year_made)
);
```

```
DROP TABLE IF EXISTS MOVIE_STAR;
CREATE TABLE MOVIE_STAR (
   full_name VARCHAR(255)
      PRIMARY KEY
 , address VARCHAR(255)
 , gener CHAR(1)
 , birthdate DATE
);
```

```
STARS_IN (
     title:      string,
     year_made:  integer,
     star_name:  string
)
```

```
MOVIE_EXECUTIVE (
     full_name: string,
     address:   string,
     cert_num:  integer,
     net_worth: integer
)
```

```
STUDIO (
     legal_name:    string,
     address:       string,
     prez_cert_num: integer
)
```

SQL I REVIEW EXERCISES…

```sql
DROP TABLE IF EXISTS MOVIE;
CREATE TABLE MOVIE (
  title VARCHAR(50)
, year_made INT
, length_mins INT
, genre VARCHAR(20)
, studio_name VARCHAR(50)
, producer_num INT
, PRIMARY KEY(title, year_made)
);
```

```sql
DROP TABLE IF EXISTS MOVIE_STAR;
CREATE TABLE MOVIE_STAR (
   full_name VARCHAR(255)
      PRIMARY KEY
 , address VARCHAR(255)
 , gener CHAR(1)
 , birthdate DATE
);
```

```sql
DROP TABLE IF EXISTS STARS_IN;
CREATE TABLE STARS_IN (
   title VARCHAR(50)
 , year_made INT
 , star_name VARCHAR(255)
 , PRIMARY KEY (title
              , year_made
              , star_name)
);
```

```
MOVIE_EXECUTIVE (
    full_name: string,
    address:   string,
    cert_num:  integer,
    net_worth: integer
)
```

```
STUDIO (
    legal_name:    string,
    address:       string,
    prez_cert_num: integer
)
```

SQL I REVIEW EXERCISES...

```
DROP TABLE IF EXISTS MOVIE;
CREATE TABLE MOVIE (
  title VARCHAR(50)
, year_made INT
, length_mins INT
, genre VARCHAR(20)
, studio_name VARCHAR(50)
, producer_num INT
, PRIMARY KEY(title, year_made)
);
```

```
DROP TABLE IF EXISTS MOVIE_STAR;
CREATE TABLE MOVIE_STAR (
   full_name VARCHAR(255)
      PRIMARY KEY
 , address VARCHAR(255)
 , gener CHAR(1)
 , birthdate DATE
);
```

```
DROP TABLE IF EXISTS STARS_IN;
CREATE TABLE STARS_IN (
   title VARCHAR(50)
 , year_made INT
 , star_name VARCHAR(255)
 , PRIMARY KEY (title
             , year_made
             , star_name)
);
```

```
DROP TABLE IF EXISTS
  MOVIE_EXECUTIVE;
CREATE TABLE
  MOVIE_EXECUTIVE (
   full_name VARCHAR(255)
 , address VARCHAR(255)
 , cert_num INT PRIMARY KEY
 , net_worth INT
);
```

```
STUDIO (
     legal_name:    string,
     address:       string,
     prez_cert_num: integer
)
```

SQL I REVIEW EXERCISES...

```
DROP TABLE IF EXISTS MOVIE;
CREATE TABLE MOVIE (
  title VARCHAR(50)
, year_made INT
, length_mins INT
, genre VARCHAR(20)
, studio_name VARCHAR(50)
, producer_num INT
, PRIMARY KEY(title, year_made)
);
```

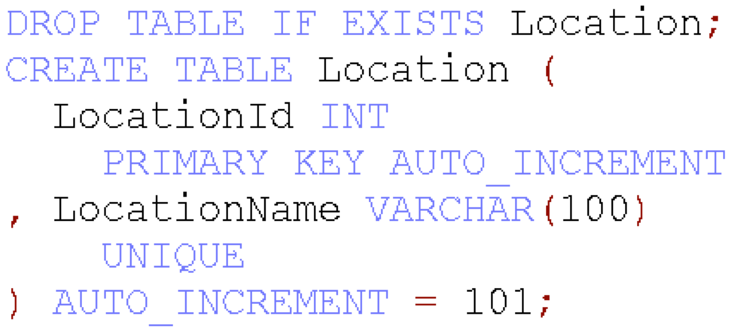
```
DROP TABLE IF EXISTS MOVIE_STAR;
CREATE TABLE MOVIE_STAR (
   full_name VARCHAR(255)
      PRIMARY KEY
 , address VARCHAR(255)
 , gener CHAR(1)
 , birthdate DATE
);
```

```
DROP TABLE IF EXISTS STARS_IN;
CREATE TABLE STARS_IN (
   title VARCHAR(50)
 , year_made INT
 , star_name VARCHAR(255)
 , PRIMARY KEY (title
              , year_made
              , star_name)
);
```

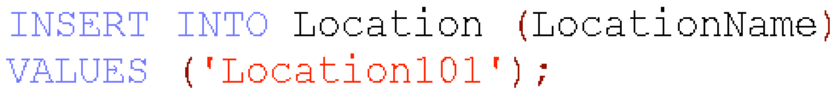
```
DROP TABLE IF EXISTS
  MOVIE_EXECUTIVE;
CREATE TABLE
  MOVIE_EXECUTIVE (
   full_name VARCHAR(255)
 , address VARCHAR(255)
 , cert_num INT PRIMARY KEY
 , net_worth INT
);
```

```
DROP TABLE IF EXISTS STUDIO;
CREATE TABLE STUDIO (
   legal_name VARCHAR(100)
 , address VARCHAR(255)
 , prez_cert_num INT PRIMARY KEY
);
```

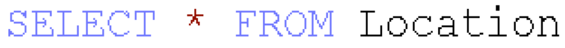
Maria DB Auto Increment



```
DROP TABLE IF EXISTS Location;
CREATE TABLE Location (
  LocationId INT
    PRIMARY KEY AUTO_INCREMENT
, LocationName VARCHAR(100)
    UNIQUE
) AUTO_INCREMENT = 101;
```



```
INSERT INTO Location (LocationName)
VALUES ('Location101');
```



```
SELECT * FROM Location
```

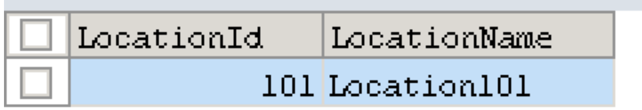
Maria DB Auto Increment

```
DROP TABLE IF EXISTS Location;
CREATE TABLE Location (
  LocationId INT
    PRIMARY KEY AUTO_INCREMENT
, LocationName VARCHAR(100)
    UNIQUE
) AUTO_INCREMENT = 101;
```

```
 INSERT INTO Location (LocationName)
 VALUES ('Location101');
```

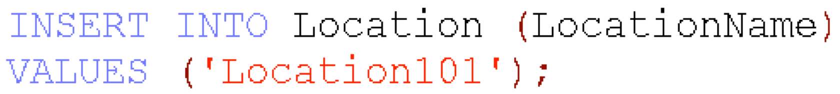
```
  SELECT * FROM Location
```

Starts at 101 like we'd expect!



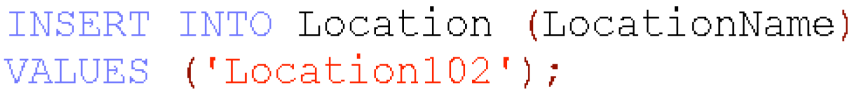
□	LocationId	LocationName
□	101	Location101

Maria DB Auto Increment

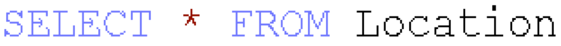


```
INSERT INTO Location (LocationName)
VALUES ('Location101');
```

<Ooops... Error, cannot insert due to unique key contraint>



```
INSERT INTO Location (LocationName)
VALUES ('Location102');
```



```
SELECT * FROM Location
```

What do you expect will be the results?

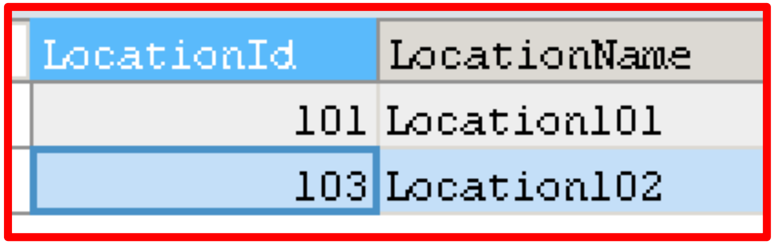
Maria DB Auto Increment

```
INSERT INTO Location (LocationName)
VALUES ('Location101');
```

<Ooops… Error, cannot insert due to unique key contraint>

```
INSERT INTO Location (LocationName)
VALUES ('Location102');
```

```
SELECT * FROM Location
```



LocationId	LocationName
101	Location101
103	Location102

Insert the following Tuples into Movie

title	year_made	length_mins	genre	studio_name	producer_num
Galaxy Quest	1999	104	comedy	DreamWorks	67890
Star Wars	1977	124	sciFi	Fox	12345
Wayne’s World	1992	95	comedy	NULL	NULL

Insert the following Tuples into Movie

title	year_made	length_mins	genre	studio_name	producer_num
Galaxy Quest	1999	104	comedy	DreamWorks	67890
Star Wars	1977	124	sciFi	Fox	12345
Wayne’s World	1992	95	comedy	NULL	NULL

```
INSERT INTO MOVIE VALUES
  ('Galaxy Quest'  , 1999, 104, 'comedy', 'DreamWorks', 67890)
, ('Star Wars'     , 1977, 104, 'sciFi' , 'Fox'       , 12345)
, ('Wayne\'s World', 1992,  95, 'comedy',  NULL       , NULL);
```

Insert the following Tuples into Movie

title	year_made	length_mins	genre	studio_name	producer_num
The Matrix	1999	150	NULL	NULL	NULL

Insert the following Tuples into Movie

title	year_made	length_mins	genre	studio_name	producer_num
The Matrix	1999	150	NULL	NULL	NULL

```
INSERT INTO MOVIE (year_made, length_mins, title)
          VALUES (1999     , 150        , 'The Matrix');
```

UNIVERSITY *of* WASHINGTON

JOINS

Implicit Joins vs Explicit Joins

```
SELECT *
  FROM t1, t2
 WHERE t1.b = t2.b
```

$\sigma_{t1.b\ =\ t2.b}(t1\ x\ t2)$

```
SELECT *
  FROM t1
  JOIN t2
    ON t2.b = t2.b
```

$t1 \bowtie_{t1.b=t2.b} t2$

JOIN EXAMPLES

```
MOVIE (title, year_made, length_mins_genre, studio_name, producer_num)
MOVIE_STAR (name, address, gender, birthdate)
STARS_IN (title, year_made, star_name)
MOVIE_EXECUTIVE (full_name, address, cert_num, net_worth)
STUDIO (legal_name, address, prez_cert_num)
```

Write the following queries in SQL:

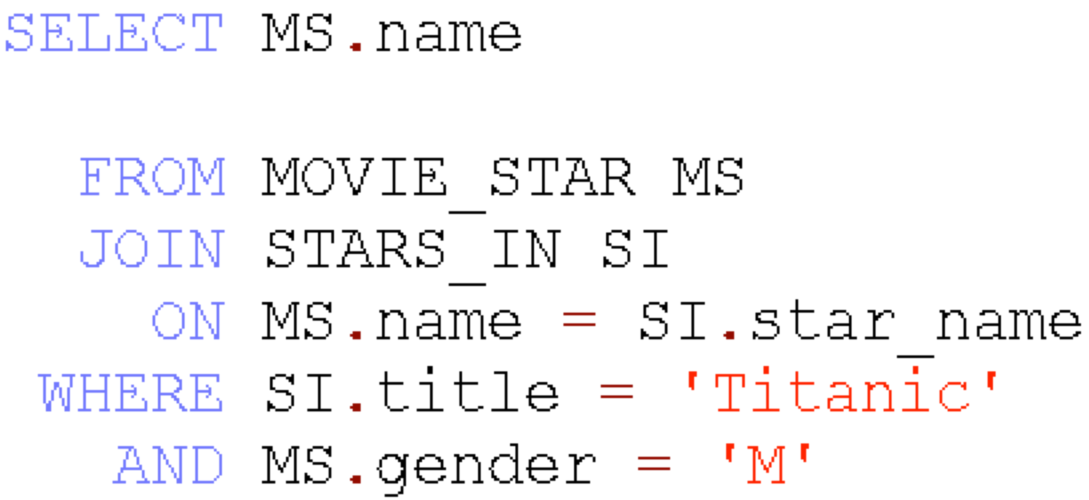
Who were the male stars in *Titanic*?

Which stars appeared in movies produced by MGM in 1995?

Who is the president of MGM studios?

JOIN EXAMPLES

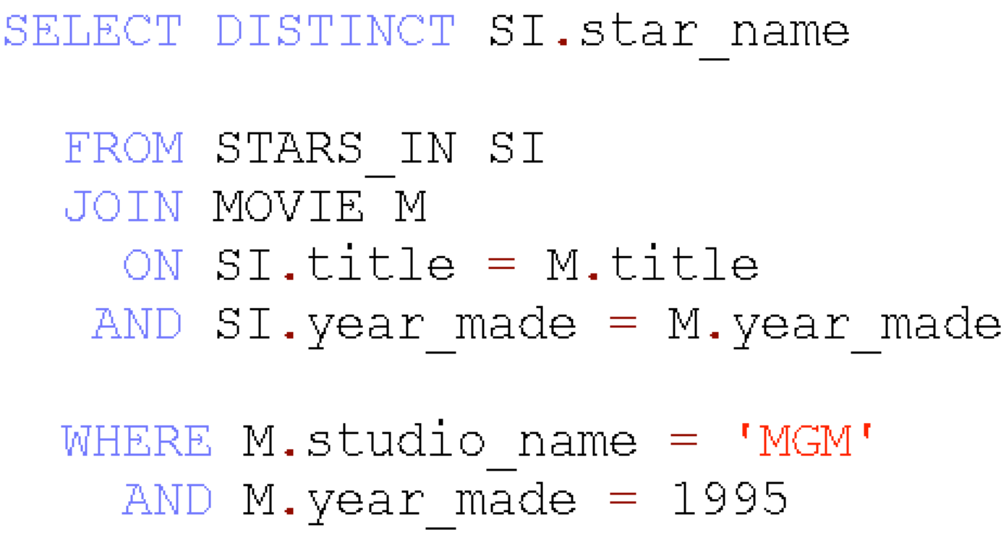
Who were the male stars in *Titanic*?



```
SELECT MS.name

  FROM MOVIE_STAR MS
  JOIN STARS_IN SI
    ON MS.name = SI.star_name
 WHERE SI.title = 'Titanic'
   AND MS.gender = 'M'
```

JOIN EXAMPLES

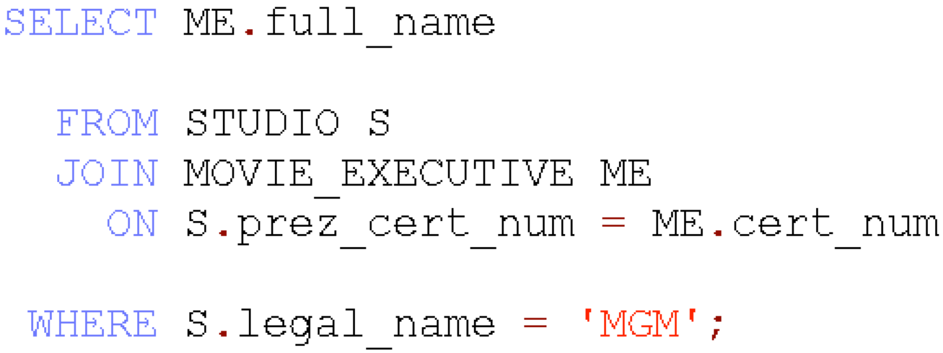
Which stars appeared in movies produced by MGM in 1995?



```
SELECT DISTINCT SI.star_name

  FROM STARS_IN SI
  JOIN MOVIE M
    ON SI.title = M.title
   AND SI.year_made = M.year_made

  WHERE M.studio_name = 'MGM'
    AND M.year_made = 1995
```

JOIN EXAMPLES

Who is the president of MGM studios?



```
SELECT ME.full_name

  FROM STUDIO S
  JOIN MOVIE_EXECUTIVE ME
    ON S.prez_cert_num = ME.cert_num

 WHERE S.legal_name = 'MGM';
```

UNIVERSITY *of* WASHINGTON

Subqueries

What are Subqueries

- Queries that are embedded into other queries are called “Subqueries”
- Subqueries can sometimes run many levels deep
- Use sparingly, can really decrease performance if misused

What can subqueries return?

- A single “scalar” value:
  - (e.g. SELECT COUNT(*) FROM MOVIE WHERE year = 1995)

- A relation of a single column, multiple rows
  - (e.g. SELECT title FROM MOVIE WHERE year = 1995)

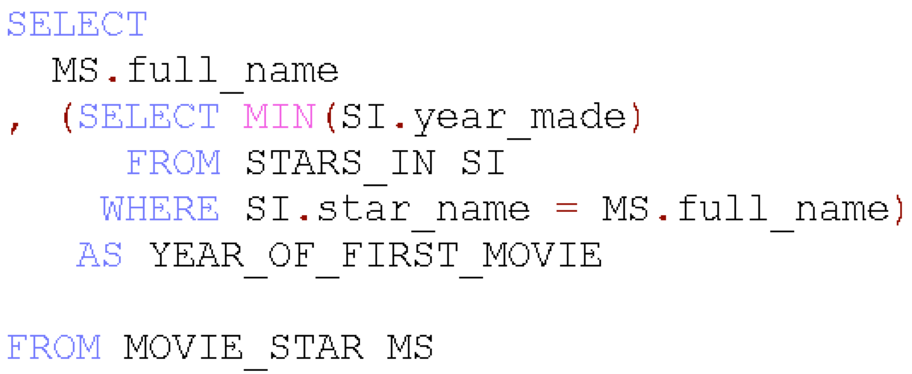
- A relation of multiple rows and columns
  - (e.g. SELECT name, studio FROM MOVIE)

Where are subqueries used?

> Where they are used depends on what they return

> Scalar Return:

  – SELECT CLAUSE

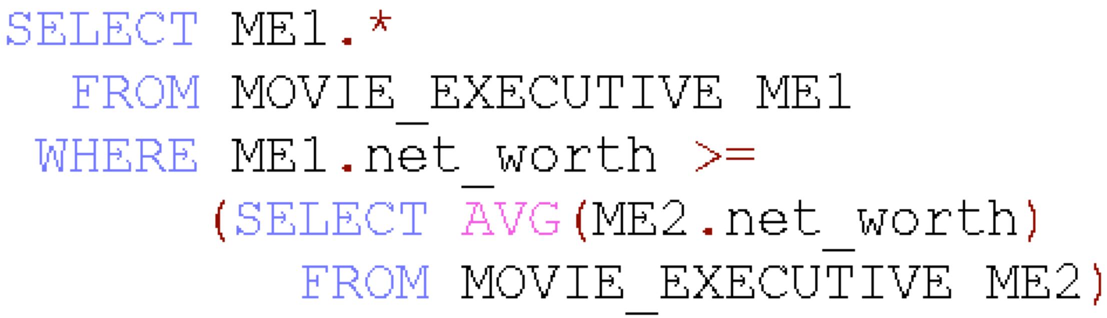


```
SELECT
  MS.full_name
,  (SELECT MIN(SI.year_made)
     FROM STARS_IN SI
    WHERE SI.star_name = MS.full_name)
   AS YEAR_OF_FIRST_MOVIE

FROM MOVIE_STAR MS
```

Where are subqueries used?

- Where they are used depends on what they return

- Scalar Return:
  - WHERE CLAUSE

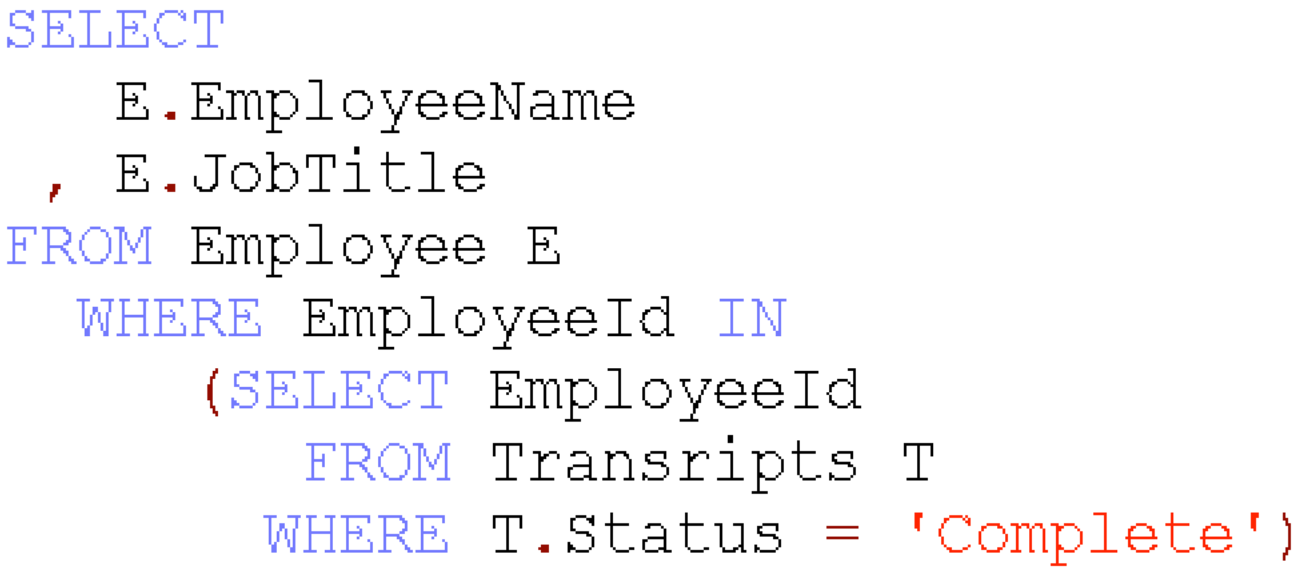


```
SELECT ME1.*
  FROM MOVIE_EXECUTIVE ME1
 WHERE ME1.net_worth >=
      (SELECT AVG(ME2.net_worth)
         FROM MOVIE_EXECUTIVE ME2)
```

Where are subqueries used?

- > Where they are used depends on what they return

- > Single Column Return

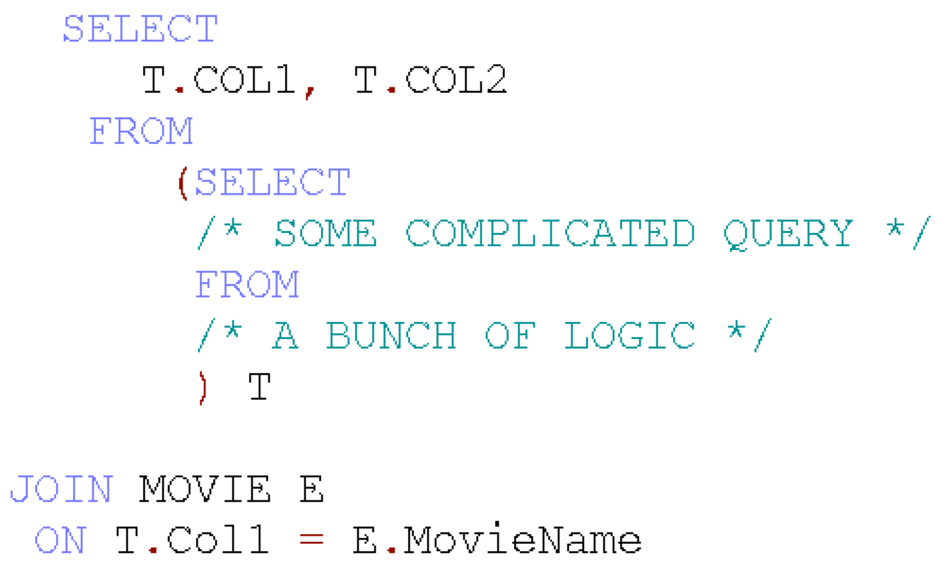


```
SELECT
   E.EmployeeName
 , E.JobTitle
FROM Employee E
  WHERE EmployeeId IN
     (SELECT EmployeeId
        FROM Transripts T
       WHERE T.Status = 'Complete')
```

Where are subqueries used?

- Where they are used depends on what they return

- Multiple Rows, Multiple Columns
  - FROM / JOIN



```
  SELECT
     T.COL1, T.COL2
   FROM
      (SELECT
       /* SOME COMPLICATED QUERY */
       FROM
       /* A BUNCH OF LOGIC */
       ) T

JOIN MOVIE E
 ON T.Col1 = E.MovieName
```

Operators that Deal with Relations

> **EXISTS**

The subquery returns at least one record

> **IN**

Similar to a long “or” statement

– Age in (Subquery)… Age = 1 or Age = 2, or Age = 5)

> **ANY**

The condition holds for a single value

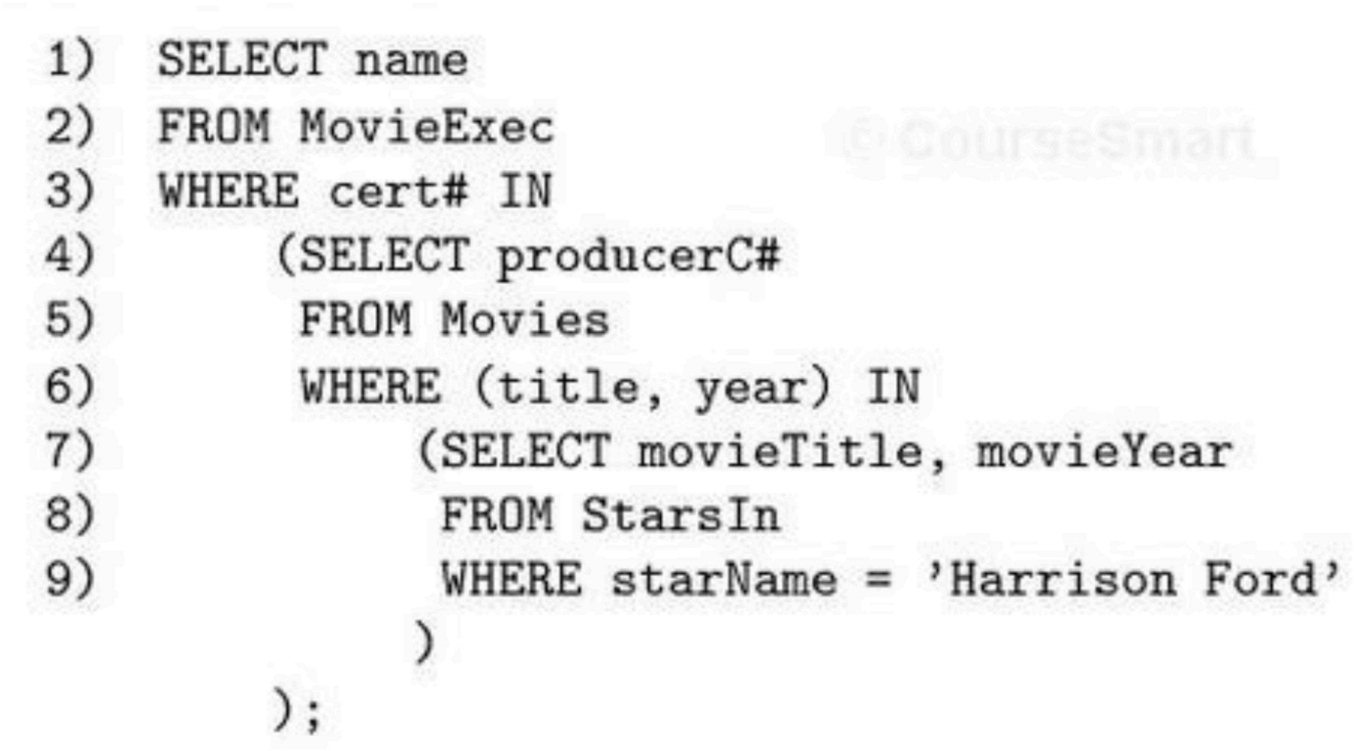
– Age > ANY (Subquery)

> **ALL**

The condition holds for all values

– Age > ALL (Subquery)

More Examples (IN)



```
SELECT name
FROM MovieExec
WHERE cert# IN
    (SELECT producerC#
     FROM Movies
     WHERE (title, year) IN
         (SELECT movieTitle, movieYear
          FROM StarsIn
          WHERE starName = 'Harrison Ford'
         )
    );
```

More Examples (EXISTS)

List the full cast from movies who have at least one movie executive starring in the movie.

```
SELECT *
FROM STARS_IN SI

WHERE EXISTS
     (SELECT ME.full_name
        FROM MOVIE_EXECUTIVE ME
        JOIN STARS_IN SI2
          ON ME.full_NAME = SI2.full_name
       WHERE SI.title     = SI2.title
         AND SI.year_made = SI2.year_made)
```

More Examples (ANY | ALL)

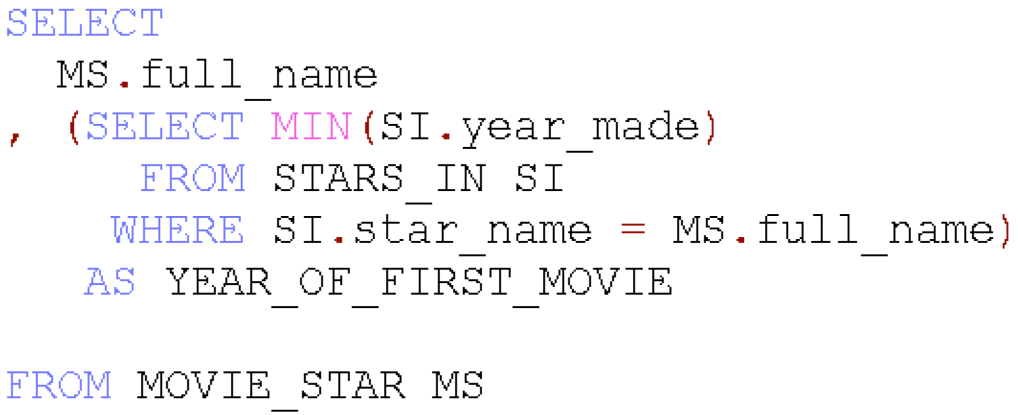
Find the brand and wattages of lightbulbs that are not the brightest. Second, find the ones that are

```
SELECT
  B.Brand
, B.Wattage as "NotTheBrightestBulbInTheBox"

FROM LightBulbs B

WHERE B.Wattage < ANY (SELECT B2.Wattage
                         FROM LightBulbs B2)


SELECT
  B.Brand
, B.Wattage as "TheBrightestBulbInTheBox"

FROM LightBulbs B

WHERE B.Wattage >= ALL (SELECT B2.Wattage
                         FROM LightBulbs B2)
```

Correlated Queries

Some subqueries can need to be executed many times, essentially once for each tuple returned by the outer query.

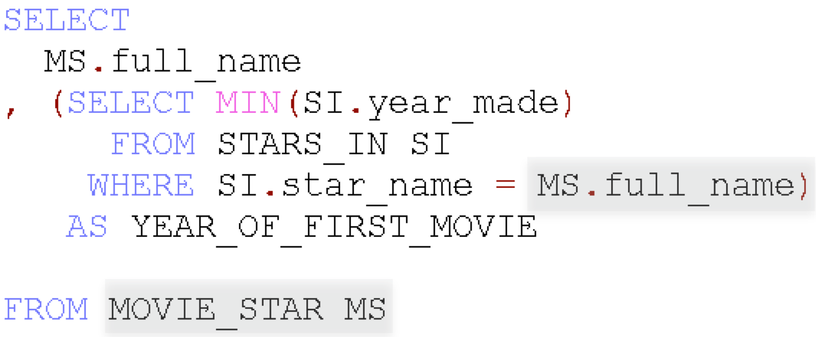
Queries that contain these types of subqueries are called **Correlated Queries**.



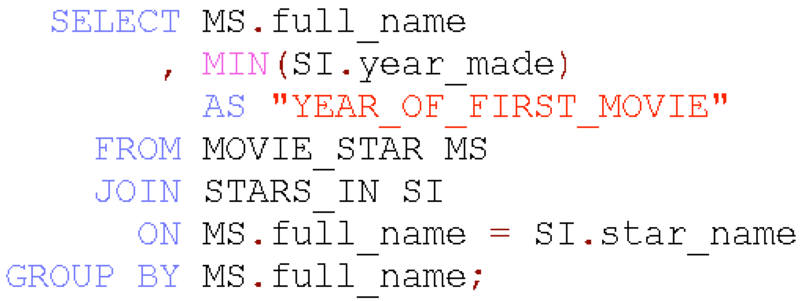
```
SELECT
  MS.full_name
,  (SELECT MIN(SI.year_made)
     FROM STARS_IN SI
    WHERE SI.star_name = MS.full_name)
   AS YEAR_OF_FIRST_MOVIE

FROM MOVIE_STAR MS
```

Correlated Queries

Can sometimes be rewritten, sometimes not. Rarely should be used on queries that return significantly large result sets.



```
SELECT
  MS.full_name
, (SELECT MIN(SI.year_made)
     FROM STARS_IN SI
    WHERE SI.star_name = MS.full_name)
   AS YEAR_OF_FIRST_MOVIE

FROM MOVIE_STAR MS
```



```
  SELECT MS.full_name
       , MIN(SI.year_made)
         AS "YEAR_OF_FIRST_MOVIE"
    FROM MOVIE_STAR MS
    JOIN STARS_IN SI
      ON MS.full_name = SI.star_name
GROUP BY MS.full_name;
```

UNIVERSITY of WASHINGTON

OUTER JOINS AND CROSS JOINS

Outer Joins

- Combine two tables on a given condition.
- Dangling tuples will remain in the result.
  - From which table depends on the type of outer join
    - FULL: Dangling Tuples from both tables remain
    - LEFT: Dangling Tuples from the LEFT table remain
    - RIGHT: Dangling Tuples from the RIGHT table remain

Outer Joins

- The ”LEFT” table is to the “LEFT” of the JOIN
- THE “RIGHT” table is to the “RIGHT” of the JOIN

```
SELECT *
FROM left_table _____ OUTER JOIN right_table
```

Outer Joins Examples: LEFT OUTER

T1

A	B
1	2
2	3
3	2
4	6

T2

B	C
1	Shake It All About
2	Left Hand In
3	Left Hand Out
4	What's It All About?

```
SELECT *
FROM T1
LEFT OUTER JOIN T2
  ON T1.B = T2.B
```

Outer Joins Examples: LEFT OUTER

T1

A	B
1	2
2	3
3	2
4	6

T2

B	C
1	Shake It All About
2	Left Hand In
3	Left Hand Out
4	What's It All About?

```
SELECT *
FROM T1
LEFT OUTER JOIN T2
  ON T1.B = T2.B
```

A	T1.B	T2.B	C
1	2	2	Left Hand In
2	2	2	Left Hand Out
3	2	2	Left Hand In
4	6	*NULL*	*NULL*

Outer Joins Examples: RIGHT OUTER

T1

A	B
1	2
2	3
3	2
4	6

T2

B	C
1	Shake It All About
2	Left Hand In
3	Left Hand Out
4	What's It All About?

```
SELECT *
FROM T1
RIGHT OUTER JOIN T2
  ON T1.B = T2.B
```

Outer Joins Examples: RIGHT OUTER

T1

A	B
1	2
2	3
3	2
4	6

T2

B	C
1	Shake It All About
2	Left Hand In
3	Left Hand Out
4	What's It All About?

```
SELECT *
FROM T1
RIGHT OUTER JOIN T2
  ON T1.B = T2.B
```

A	T1.B	T2.B	C
NULL	NULL	1	Shake it All About
1	2	2	Left Hand In
3	2	2	Left Hand In
2	3	3	Left Hand Out
NULL	NULL	4	What's It All About

Outer Joins Examples: FULL OUTER

T1

A	B
1	2
2	3
3	2
4	6

T2

B	C
1	Shake It All About
2	Left Hand In
3	Left Hand Out
4	What's It All About?

```
SELECT *
FROM T1
FULL OUTER JOIN T2
  ON T1.B = T2.B
```

Outer Joins Examples: FULL OUTER

T1

A	B
1	2
2	3
3	2
4	6

T2

B	C
1	Shake It All About
2	Left Hand In
3	Left Hand Out
4	What's It All About?

```
SELECT *
FROM T1
FULL OUTER JOIN T2
  ON T1.B = T2.B
```

A	T1.B	T2.B	C
1	2	2	Left Hand In
2	3	3	Left Hand Out
3	2	2	Left Hand In
4	6	*NULL*	*NULL*
NULL	*NULL*	1	Shake it All About
NULL	*NULL*	4	What's It All About?

Cross Joins (Cartesian Product)

> Every tuple of the left table is paired with every tuple of the right table. |R|x|S| = |R x S|

R

A	B
1	2
3	4

S

C	D
5	6
7	8

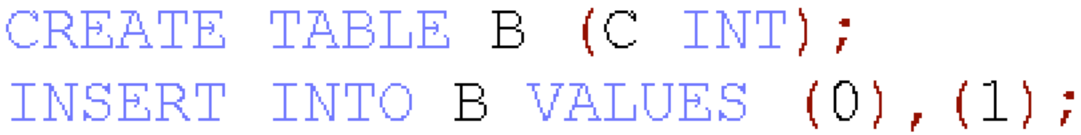
A	B	C	D
1	2	5	6
1	2	7	8
3	4	5	6
3	4	7	8

Cross Join Syntax

```
SELECT *
FROM R
CROSS JOIN S
```

```
SELECT *
FROM R,S
```

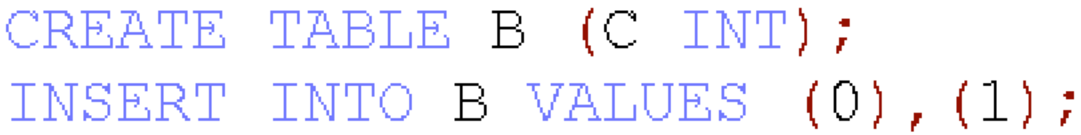
Fun with Cross Joins



```
CREATE TABLE B (C INT);
INSERT INTO B VALUES (0),(1);
```

C
0
1

Fun with Cross Joins

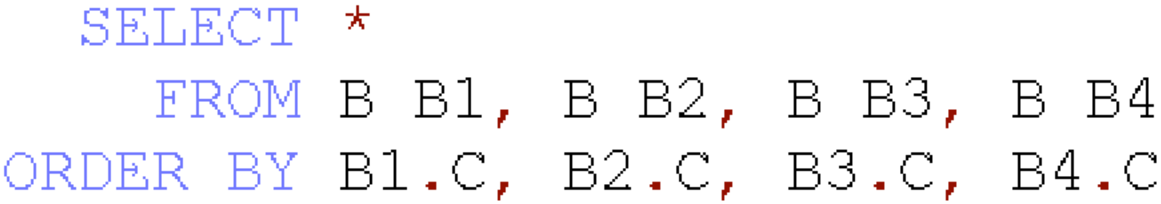


```
CREATE TABLE B (C INT);
INSERT INTO B VALUES (0),(1);
```

C
0
1

WHAT WILL B x B x B x B LOOK LIKE?

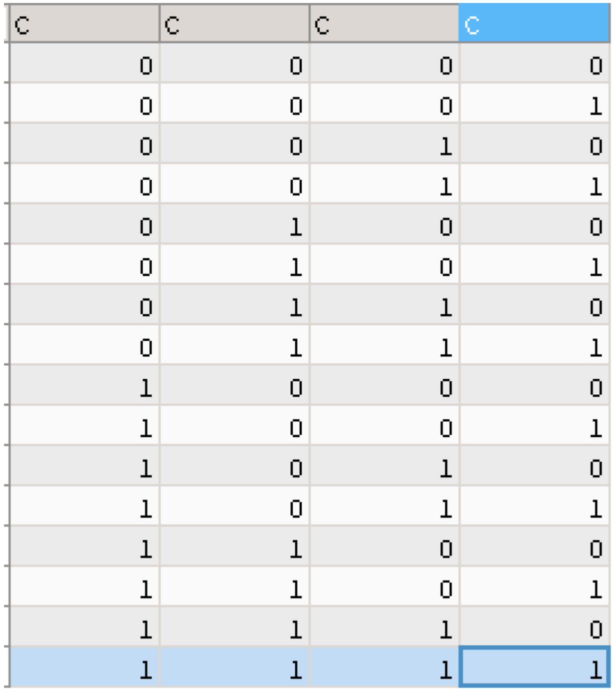
Fun with Cross Joins



```
  SELECT *
    FROM B B1, B B2, B B3, B B4
ORDER BY B1.C, B2.C, B3.C, B4.C
```

Fun with Cross Joins

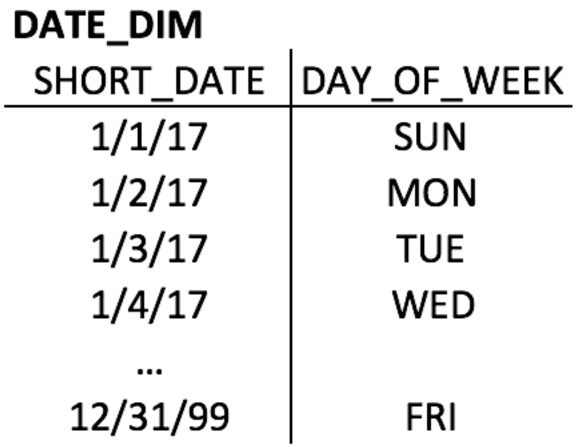
```
  SELECT *
    FROM B B1, B B2, B B3, B B4
ORDER BY B1.C, B2.C, B3.C, B4.C
```



C	C	C	C
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

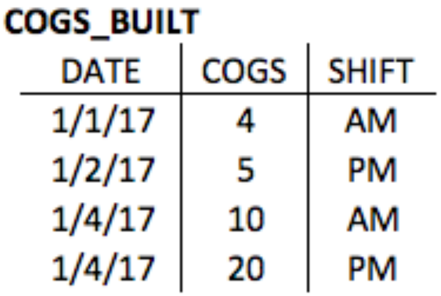
Use Case: For Each Day

Provide a list that includes every day, every shift, how many cogs were made.



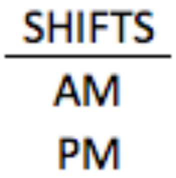
DATE_DIM

SHORT_DATE	DAY_OF_WEEK
1/1/17	SUN
1/2/17	MON
1/3/17	TUE
1/4/17	WED
…	
12/31/99	FRI

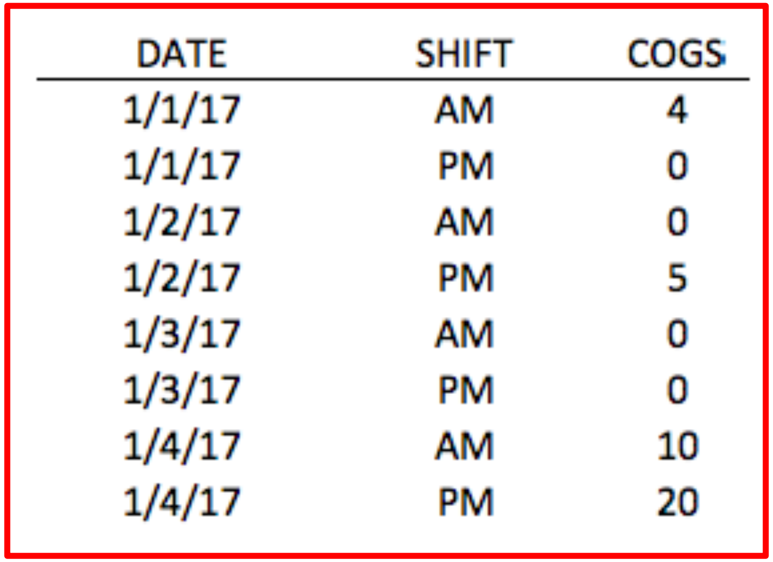


COGS_BUILT

DATE	COGS	SHIFT
1/1/17	4	AM
1/2/17	5	PM
1/4/17	10	AM
1/4/17	20	PM



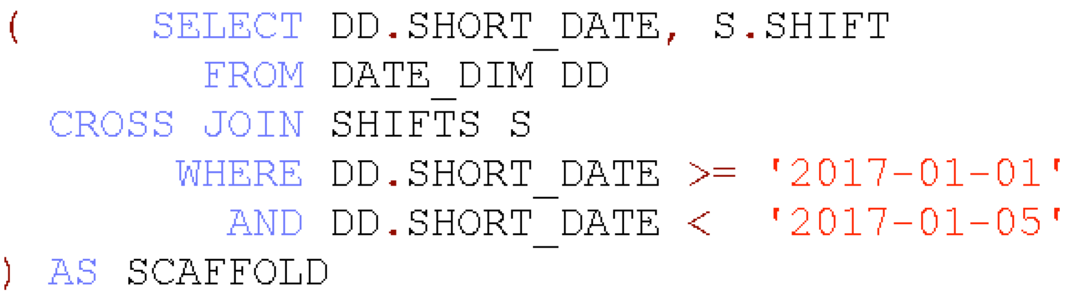
SHIFTS
AM
PM



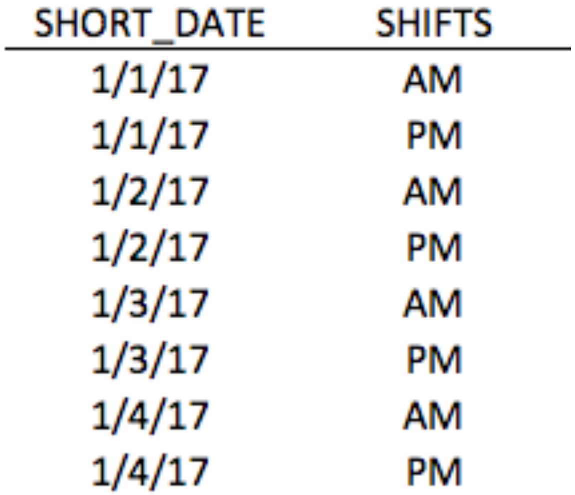
DATE	SHIFT	COGS
1/1/17	AM	4
1/1/17	PM	0
1/2/17	AM	0
1/2/17	PM	5
1/3/17	AM	0
1/3/17	PM	0
1/4/17	AM	10
1/4/17	PM	20

Use Case: For Each Day

START WITH A CROSS JOIN TO BUILD YOUR SCAFFOLD



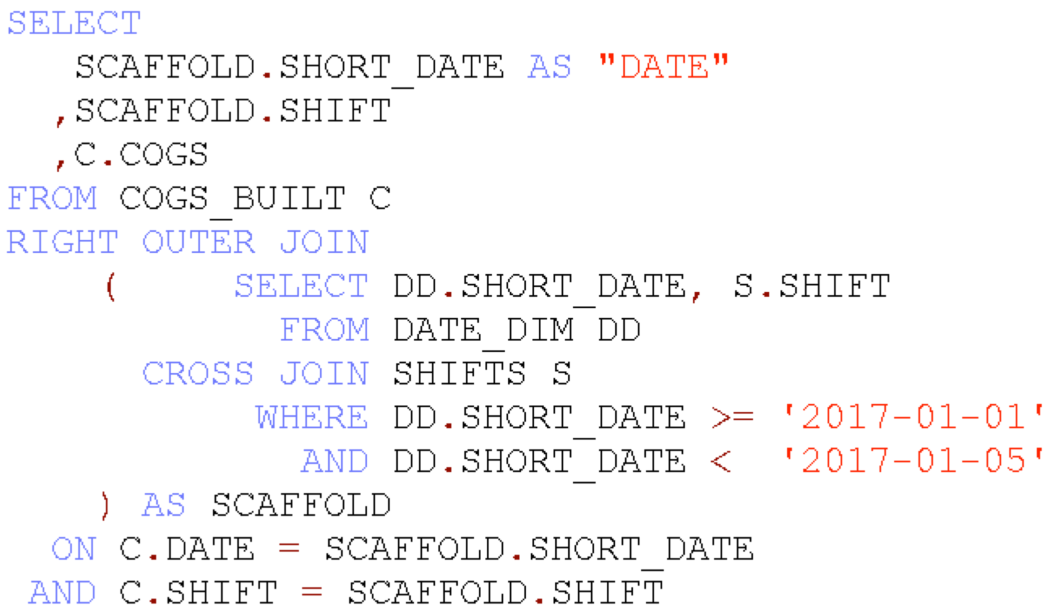
```
(     SELECT DD.SHORT_DATE, S.SHIFT
        FROM DATE_DIM DD
  CROSS JOIN SHIFTS S
       WHERE DD.SHORT_DATE >= '2017-01-01'
         AND DD.SHORT_DATE <  '2017-01-05'
) AS SCAFFOLD
```



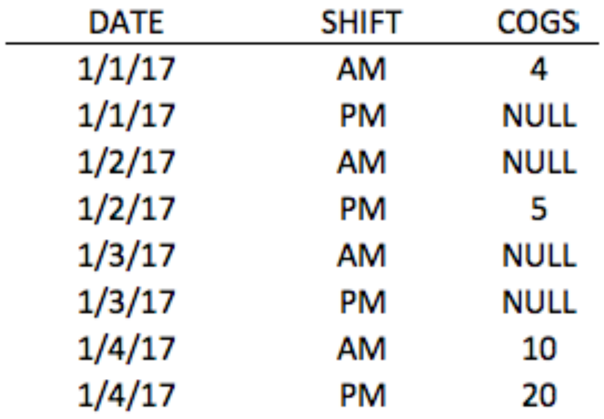
SHORT_DATE	SHIFTS
1/1/17	AM
1/1/17	PM
1/2/17	AM
1/2/17	PM
1/3/17	AM
1/3/17	PM
1/4/17	AM
1/4/17	PM

Use Case: For Each Day

THEN OUTER JOIN COGS_BUILT TO THE SCAFFOLD TO GET THE COG COUNTS



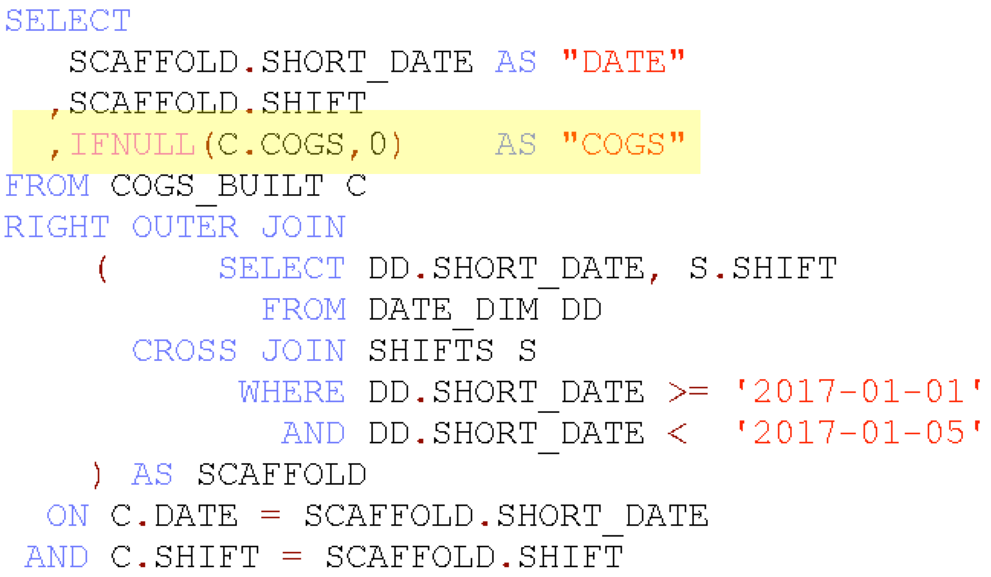
```
SELECT
   SCAFFOLD.SHORT_DATE AS "DATE"
  ,SCAFFOLD.SHIFT
  ,C.COGS
FROM COGS_BUILT C
RIGHT OUTER JOIN
    (      SELECT DD.SHORT_DATE, S.SHIFT
             FROM DATE_DIM DD
      CROSS JOIN SHIFTS S
            WHERE DD.SHORT_DATE >= '2017-01-01'
              AND DD.SHORT_DATE <  '2017-01-05'
    ) AS SCAFFOLD
  ON C.DATE = SCAFFOLD.SHORT_DATE
 AND C.SHIFT = SCAFFOLD.SHIFT
```



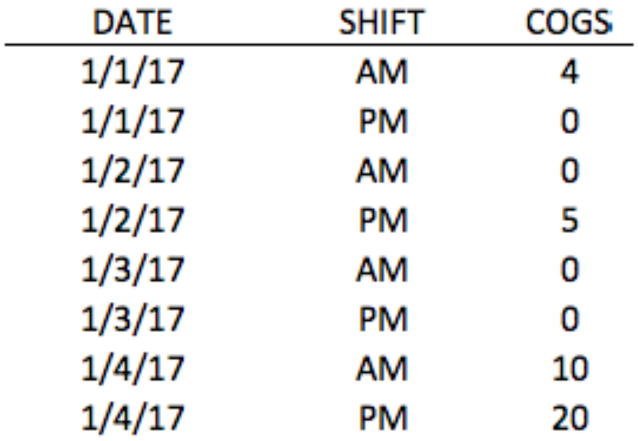
DATE	SHIFT	COGS
1/1/17	AM	4
1/1/17	PM	NULL
1/2/17	AM	NULL
1/2/17	PM	5
1/3/17	AM	NULL
1/3/17	PM	NULL
1/4/17	AM	10
1/4/17	PM	20

Use Case: For Each Day

LASTLY WE HAVE TO HANDLE NULLS! IN MARIADB, USE IFNULL(value,value_if_null)



```
SELECT
   SCAFFOLD.SHORT_DATE AS "DATE"
  ,SCAFFOLD.SHIFT
  ,IFNULL(C.COGS,0)    AS "COGS"
FROM COGS_BUILT C
RIGHT OUTER JOIN
    (     SELECT DD.SHORT_DATE, S.SHIFT
            FROM DATE_DIM DD
      CROSS JOIN SHIFTS S
           WHERE DD.SHORT_DATE >= '2017-01-01'
             AND DD.SHORT_DATE <  '2017-01-05'
    ) AS SCAFFOLD
  ON C.DATE = SCAFFOLD.SHORT_DATE
 AND C.SHIFT = SCAFFOLD.SHIFT
```



DATE	SHIFT	COGS
1/1/17	AM	4
1/1/17	PM	0
1/2/17	AM	0
1/2/17	PM	5
1/3/17	AM	0
1/3/17	PM	0
1/4/17	AM	10
1/4/17	PM	20

UNIVERSITY *of* WASHINGTON

Natural Joins

Natural Joins.   Possible in SQL

Matches on columns with identical names.
Depends on a well formed schema.

```
     SELECT *
       FROM Employee
NATURAL JOIN TrainingStatus
```

A look at all of the joins...

```
FROM R, S
FROM R NATURAL JOIN S
FROM R NATURAL [LEFT|RIGHT|FULL] OUTER JOIN S

FROM R CROSS JOIN S

FROM R JOIN S ON...
FROM R [LEFT|RIGHT|FULL] OUTER JOIN S ON...
```

UNIVERSITY *of* WASHINGTON

Grouping & Aggregation

Aggregation

SUM, AVG, MIN, MAX, COUNT

```
SELECT COUNT(*) FROM MOVIE

SELECT COUNT(title) FROM MOVIE

SELECT COUNT(DISTINCT star_name) FROM STARS_IN
```

Aggregation With “Group By”

```
  SELECT AVG(length_mins) as "avg_length"
       , COUNT(*)         as "movies_made"
    FROM MOVIE
GROUP BY studio_name;

SELECT year_made
     , count(*) as "movies_made”
  FROM MOVIE
GROUP BY year_made;
```

Aggregation With “Group By”

```
SELECT full_name
  FROM STARS_IN
 GROUP BY full_name
```

EXACTLY THE SAME RESULT AS

```
SELECT DISTINCT full_name
  FROM STARS_IN
```

Aggregate Attributes & Grouping Attributes

Why will the following query fail to return the correct results?

```
SELECT studio, year_made, count(*) as release_count
FROM MOVIE
GROUP BY year_made
```

What do we have to change to get it to run?

Aggregate Attributes & Grouping Attributes

Why will the following query fail to return the correct results?

```
SELECT studio, year_made, count(*) as release_count
FROM MOVIE
GROUP BY year_made
```

What do we have to change to get it to run?

```
SELECT studio, year_made, count(*) as release_count
FROM MOVIE
GROUP BY year_made, studio
```

NULLS in Aggregation

- NULL is (in general) ignored by aggregation
  - AVG(1,null,2,3) = (1+2+3)/3 = 2
- COUNT(column) returns the number of non NULL values in that column
- COUNT(*) is the number of tuples / rows regardless of nulls

- READ DOCUMENTATION OF YOUR DBMS
  - Special cases exist when entire column or entire row is null
    - MariaDB: count(null) = 0, sum(null) = null

HAVING
(Condition applied after Grouping)

Having is a “WHERE” condition that occurs AFTER aggregation takes place.

Example: get movies that cast more than 2 movie stars

```
SELECT title, year_made
  FROM STARS_IN

GROUP BY title, year_made

HAVING COUNT(*) > 2
```

HAVING (Condition applied after Grouping)

Having is a “WHERE” condition that occurs AFTER aggregation takes place.

Example: get movies that cast more than 2 movie stars

```
SELECT title, year_made
  FROM STARS_IN

GROUP BY title, year_made

HAVING COUNT(*) > 2
```

```
SELECT T.title, T.year_made
FROM
 ( SELECT title
        , year_made
        , count(*) as movie_count
     FROM STARS_IN SI2
 GROUP BY title, year_made
 ) as T

WHERE T.movie_count > 2
```

Additional Examples

Get movies from Fox that cast more than 2 movie stars

Additional Examples

Get movies from Fox that cast more than 2 movie stars

```
SELECT SI.title, SI.year_made

  FROM STARS_IN SI
  JOIN MOVIE M
    ON SI.title = M.title
   AND SI.year_made = M.year_made

 WHERE M.studio_name = 'Fox'

GROUP BY SI.title, SI.year_made

HAVING COUNT(*) > 2
```

Additional Examples

Get the average length of movies from each studio that has produced at least 3 movies.

Additional Examples

Get the average length of movies from each studio that has produced at least 3 movies.

```
  SELECT studio_name
       , avg(length_mins) as avg_len
    FROM MOVIE
GROUP BY studio_name
  HAVING count(*) >= 3
```

Order By

- The last statement executed (typically)
- Often not allowed in subqueries (not needed)
- Ordered columns do NOT need to be in projection
- Can include Aggregated Columns

```
ORDER BY colname [ASC|DESC], colname [ASC|DESC]...
```

Order By Examples

List movie stars in alphabetical order

```
SELECT full_name, address
FROM MOVIE_STAR
ORDER BY full_name ASC
```

Order By Examples

List movie stars in order of how many movies they've starred in (most on top, ties sort in alphabetical order)

```
  SELECT full_name, count(*) as movie_count
    FROM STARS_IN
GROUP BY full_name
ORDER BY count(*) DESC, full_name ASC
```

UNIVERSITY of WASHINGTON

Unions, Intersections, and Difference

Set Operations

- SQL provides corresponding operators that apply to the results of queries, provided those queries produce relations with the same list of attributes and attribute types.
- The keywords used are UNION, INTERSECT, and EXCEPT for U, ∩, and -, respectively.
- There is a bag union called “UNION ALL” that will not remove duplicates but rather stick one relation on top of the other.
- *Not all databases support these keywords.*

Set Operations

```
MovieStar (name, address, gender, birthdate)
MovieExec (name, address, cert#, netWorth)
Movies (title, year, length, genre, studioName, producerC#)
StarsIn (movieTitle, movieYear, starName)
```

```
SELECT name, address
  FROM MovieStar
 WHERE gender = 'F'
INTERSECT
SELECT name, address
  FROM MovieExec
 WHERE netWorth > 10000000;
```

```
SELECT name, address
  FROM MovieStar
EXCEPT
SELECT name, address
  FROM MovieExec;
```

```
SELECT title, year
  FROM Movies
UNION
SELECT movieTitle AS title
     , movieYear  AS year
FROM StarsIn;
```

UNIVERSITY *of* WASHINGTON

MANIPULATING DATA THAT EXISTS

To Modify Data

- INSERT
- UPDATE
- DELETE

INSERT

```
INSERT INTO table (column_list) VALUES
  (value_list) , (value_list)
```

```
INSERT INTO table (column_list)
  SELECT …
```

To use select statement, you must have matching tuple sizes and adhere to all contraints just as if you were using VALUES (xxx),(xxx)

DELETE

- **DELETE FROM table WHERE condition…**

```
DELETE FROM StarsIn
    WHERE movie_title = ‘Star Wars’;
```

WARNING:

REPLACE **DELETE FROM** WITH **SELECT * FROM** AS A SANITY CHECK BEFORE EXECUTING YOUR DELETE STATEMENT

DELETE

- DELETING ALL DATA FROM A TABLE:

```
DELETE FROM tablename;
```

UPDATE

UPDATE table SET <value assignment> WHERE <condition>

```
UPDATE MOVIE SET title = ‘Star Wars: A New Hope’
           WHERE title = ‘Star Wars’
             AND year = ‘1977’
```

```
UPDATE MOVIE_EXECUTIVE
   SET full_name = CONCAT(‘Pres. ‘, full_name)
 WHERE LEFT(full_name,6) != ‘Pres. ‘
```