Lancaster University

SCC201: Databases

Week5: SQL and Advanced SQL Scripts

By
Uraz C turker

Week 1	Lecture 1	Introduction to the module, Why do we need Databases? Entity Relationship Model
	Lecture 2	Entity Relationship Model (ERM)
	Lab	NO LAB
Week2	Lecture 1	Relational Model (RM)
	Lecture 2	ER to RM
	Lab	ER diagrams.
Week 3	Lecture 1	Functional Dependencies
	Lecture 2	1st, 2nd, 3rd and Boyce-Cott Normal Forms
	Lab	ER to Relational Model.
Week 4	Lecture 1	Relational Algebra
	Lecture 2	Relational Algebra
	Lab	Functional dependencies and Normal forms
Week 5	Lecture 1	SQL Scripts
	Lecture 2	Advanced SQL Scripts
	Lab	Relational algebra
Week 6	Lecture 1	JDBC
	Lecture 2	Review
	Lab	Advanced SQL Scripting
Week 7	Lecture 1	Physical Storage - record files
	Lecture 2	Storage - secondary files
	Lab	Project
Week 8	Lecture 1	Record Search - B-Trees
	Lecture 2	Search - Hashing
	Lab	Project
Week 9	Lecture 1	Concurrency - Transaction Processing (cont)
	Lecture 2	Durability of Transactions and Crash Recovery
	Lab	Project
Week 10	Lecture 1	Advanced SQL - schemas, views, access control
	Lecture 2	Review
	Lab	Project

Lancaster University

From you

Lancaster University

What aspect of the module did you find most difficult?
3NF and BCNF

-

Getting head around wither to use section or projection

Currently; sorting the FDs using the Axioms

From you

Lancaster University

What aspect of the module did you find most difficult?
3NF and BCNF
-
Getting head around wither to use section or projection
Currently; sorting the FDs using the Axioms

What can be improved to help with the module?
None
I don't know if you have any control over this but there is only one physical copy of the required reading in the library. Would be good to have more for those that prefer physical book over reading off a screen.
more examples to walk though or exercises to do for relational algebra
I think that it would be useful to release the worked solutions with the lab worksheets so that we can go through and check our answers in the lab while it is fresh and so that should we get stuck, we can check we are on the right track

From you

Lancaster University

What aspect of the module did you find most difficult?
3NF and BCNF

-

Getting head around wither to use section or projection

Currently; sorting the FDs using the Axioms

What can be improved to help with the module?
None

I don't know if you have any control over this but there is only one physical copy of the required reading in the library. Would be good to have more for those that prefer physical book over reading off a screen.

more examples to walk though or exercises to do for relational algebra

I think that it would be useful to release the worked solutions with the lab worksheets so that we can go through and check our answers in the lab while it is fresh and so that should we get stuck, we can check we are on the right track

How was the support offered by module convenor?
Extra

Extra

Enough

None

From you

Lancaster University

What aspect of the module did you find most difficult?
3NF and BCNF

-

Getting head around wither to use section or projection

Currently; sorting the FDs using the Axioms

What can be improved to help with the module?
None

I don't know if you have any control over this but there is only one physical copy of the required reading in the library. Would be good to have more for those that prefer physical book over reading off a screen.

more examples to walk though or exercises to do for relational algebra

I think that it would be useful to release the worked solutions with the lab worksheets so that we can go through and check our answers in the lab while it is fresh and so that should we get stuck, we can check we are on the right track

How was the support offered by module convenor?
Extra

Extra

Enough

None

What more support can be provided?
N/A

-

none Uraz is great love that he remembers names :)

n/a

From you

Lancaster University

What aspect of the module did you find most difficult?
3NF and BCNF

-

Getting head around wither to use section or projection

Currently; sorting the FDs using the Axioms

What can be improved to help with the module?
None

I don't know if you have any control over this but there is only one physical copy of the required reading in the library. Would be good to have more for those that prefer physical book over reading off a screen.

more examples to walk though or exercises to do for relational algebra

I think that it would be useful to release the worked solutions with the lab worksheets so that we can go through and check our answers in the lab while it is fresh and so that should we get stuck, we can check we are on the right track

Any other comments
N/A

-

Do we need to use semijoins and more as when reading the chapters said to for this week it goes into a lot more such as aggregate and grouping do we need to know this for the test in the summer and do we need to learn the definitions for each one aswell as could we be asked a question saying what does this mean

n/a

How was the support offered by module convenor?
Extra

Extra

Enough

None

What more support can be provided?
N/A

-

none Uraz is great love that he remembers names :)

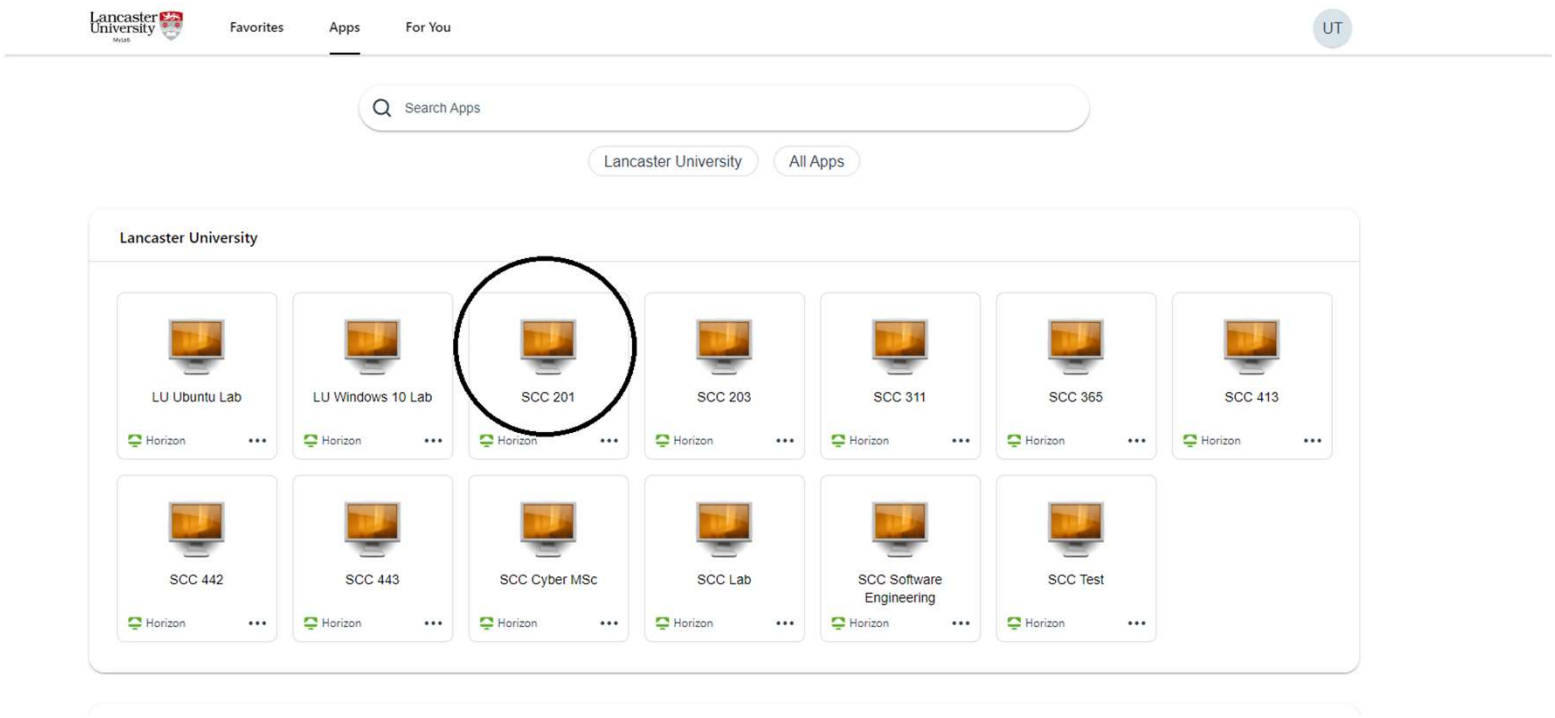
n/a

Resources

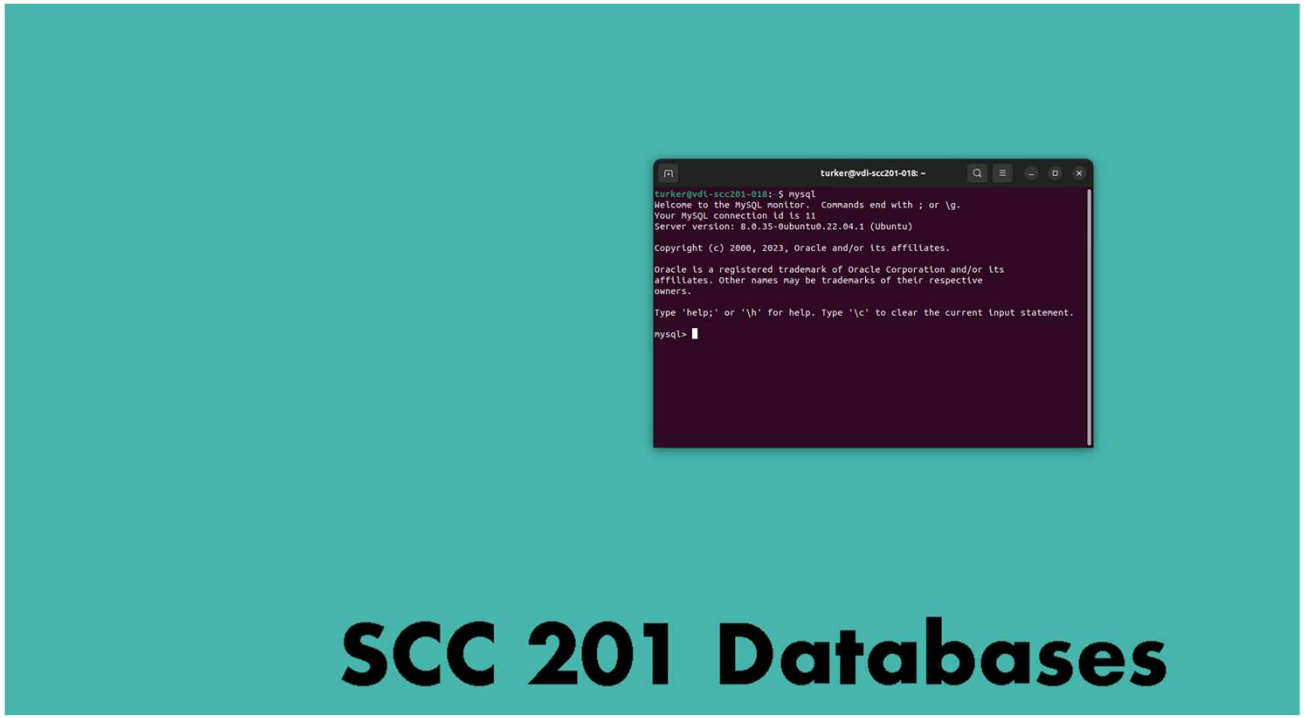
Lancaster University

https://mylab.lancaster.ac.uk

MySQL by Paul DuBois


Lancaster University
Favorites
Apps
For You
UT
Search Apps
Lancaster University
All Apps
Lancaster University
LU Ubuntu Lab
LU Windows 10 Lab
SCC 201
SCC 203
SCC 311
SCC 365
SCC 413
SCC 442
SCC 443
SCC Cyber MSc
SCC Lab
SCC Software Engineering
SCC Test
Horizon

Lancaster University



Learning outcomes

Lancaster University

This week, you will learn the basics of SQL scripting, including notations and formulations.

You will gain hands-on experience in converting ERDs and RLs into SQL scripts.

You will learn to generate nested queries and aggregate operations using SQL.

These slides and SCC.201 are based on MySQL v8; using any other versions of SQL may lead you to lose marks in coursework and exams.

The SQL Query Language

Lancaster University

- Developed by IBM (system R)
- Need for a standard since it is
- Standards:
  - SQL-86
  - SQL-89 (minor revision)
  - SQL-92 (major revision, curre
  - SQL-99 (major extensions)

```
SELECT S.rating, MIN (S.age)
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT (*) > 1
```

```
void myDisplay()
{
    game.display();
}

// Define the reshape function
void myReshape(int width, int height)
{
    game.reshape(width, height);
}

// Define the mouse click events
void myMouseClick(int button, int state, int x, int y)
{
    game.mouseClick(button, state, x, y);
}
```

Creating Relations in SQL

Lancaster University

RS:

Students(sid TEXT, name TEXT, login TEXT, age TEXT, gpa REAL);

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Shero	shero@cs	18	3.2

```
CREATE TABLE Students
        (sid TEXT,
         name TEXT,
         login TEXT,
         age INTEGER,
         gpa REAL);
```

Built-in data types varies interpreter to another.

TEXT, VARCHAR(Length), REAL, INTEGER, and BLOB are the most common data types.

Some interpreters accepts INT and INTEGER, some accepts DATA and BOOLEAN, but others don't.

Creates the Students' relation. Observe that each field's type (domain) is specified and enforced by the DBMS whenever tuples are added or modified.

CREATE *Relation's Name*(*Relational Schema Text in MySQL format*);

MySQL DATA TYPES

DATE TYPE	SPEC	DATA TYPE	SPEC
CHAR	String (0 - 255)	INT	Integer (-2147483648 to 214748-3647)
VARCHAR	String (0 - 255)	BIGINT	Integer (-9223372036854775808 to 9223372036854775807)
TINYTEXT	String (0 - 255)	FLOAT	Decimal (precise to 23 digits)
TEXT	String (0 - 65535)	DOUBLE	Decimal (24 to 53 digits)
BLOB	String (0 - 65535)	DECIMAL	"DOUBLE" stored as string
MEDIUMTEXT	String (0 - 16777215)	DATE	YYYY-MM-DD
MEDIUMBLOB	String (0 - 16777215)	DATETIME	YYYY-MM-DD HH:MM:SS
LONGTEXT	String (0 - 4294967295)	TIMESTAMP	YYYYMMDDHHMMSS
LONGBLOB	String (0 - 4294967295)	TIME	HH:MM:SS
TINYINT	Integer (-128 to 127)	ENUM	One of preset options
SMALLINT	Integer (-32768 to 32767)	SET	Selection of preset options
MEDIUMINT	Integer (-8388608 to 8388607)	BOOLEAN	TINYINT(1)

Creating Relations in SQL

Lancaster University

As another example, the Enrolled table holds information about students' courses and grades.

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Shero	shero@cs	18	3.2

Students

```
CREATE TABLE Enrolled
    (sid TEXT,
     cid TEXT,
     grade TEXT);
```

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Enrolled

Recall Keys

id	name	surname	Phone number	gpa
53666	Cherry	Hardguy	123456	3.4
53688	Cherry	Goodgirl	234567	3.4

Lancaster University

Recall Keys

id	name	surname	Phone number	gpa
53666	Cherry	Hardguy	123456	3.4
53688	Cherry	Goodgirl	234567	3.4

Lancaster University

- Primary Key: A set of attributes that can uniquely identify tuples in a given relation.
  - PK: {name,surname}
-
  -
-
  -

Recall Keys

id	name	surname	Phone number	gpa
53666	Cherry	Hardguy	123456	3.4
53688	Cherry	Goodgirl	234567	3.4

Lancaster University

- Primary Key: A set of attributes that can uniquely identify tuples in a given relation.
  - PK: {name,surname}
- Candidate Key: A set of set of attributes that can be selected as a Primary Key.
  - CK:{{id}{phone number}{name,surname}}
-
  -

Recall Keys

id	name	surname	Phone number	gpa
53666	Cherry	Hardguy	123456	3.4
53688	Cherry	Goodgirl	234567	3.4

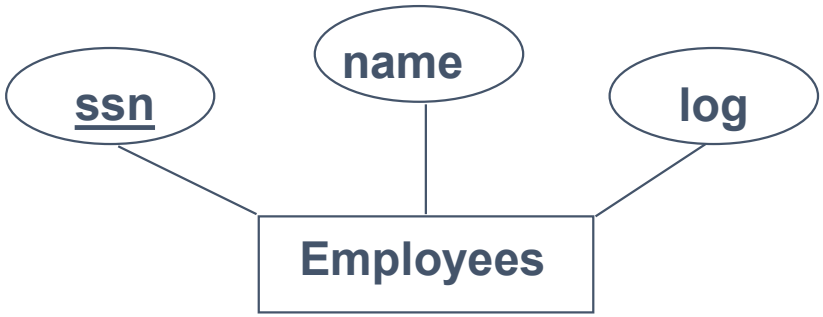
Lancaster University

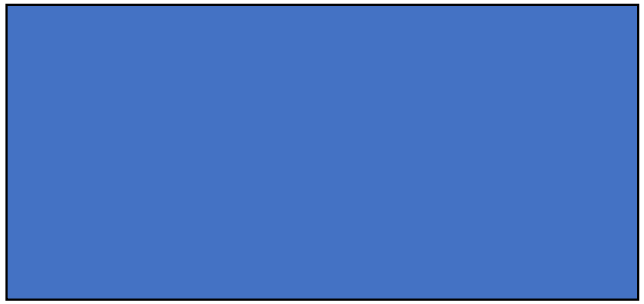
- Primary Key: A set of attributes that can uniquely identify tuples in a given relation.
  - PK: {name,surname}
- Candidate Key: A set of set of attributes that can be selected as a Primary Key.
  - CK:{{id}{phone number}{name,surname}}
- Super Key: A set of attributes having the Primary Key.
  - SK:{name,surname,gpa}

Logical DB Design: ER to Relational

Lancaster University

Entity sets to tables.

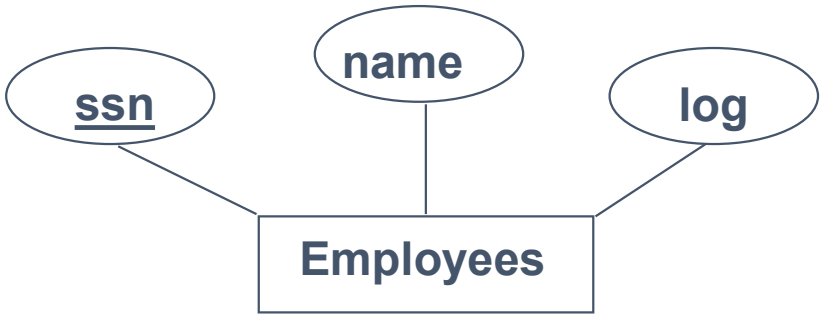




Logical DB Design: ER to Relational

Lancaster University

Entity sets to tables.

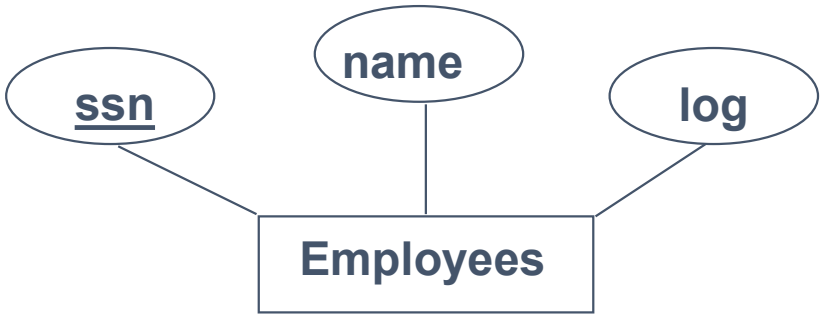



```
CREATE TABLE Employees
    (ssn VARCHAR(11),
    name VARCHAR(20),
    log  INTEGER,
```

Logical DB Design: ER to Relational

Lancaster University

Entity sets to tables.

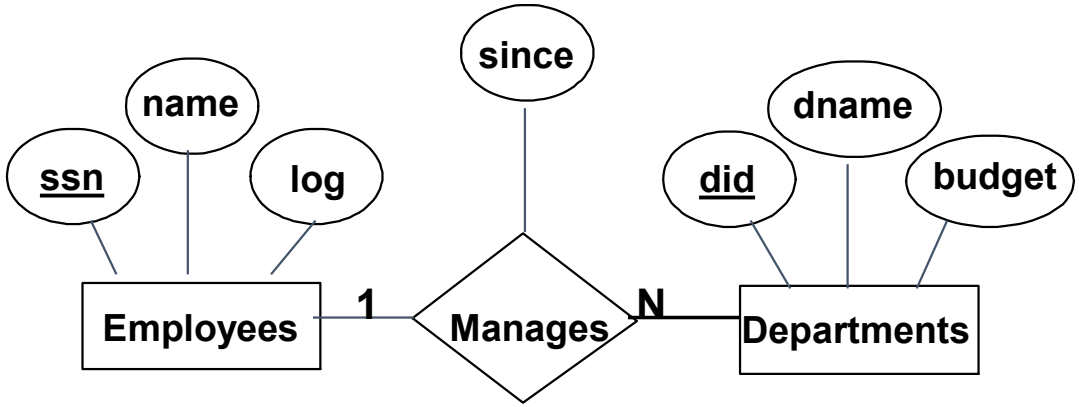


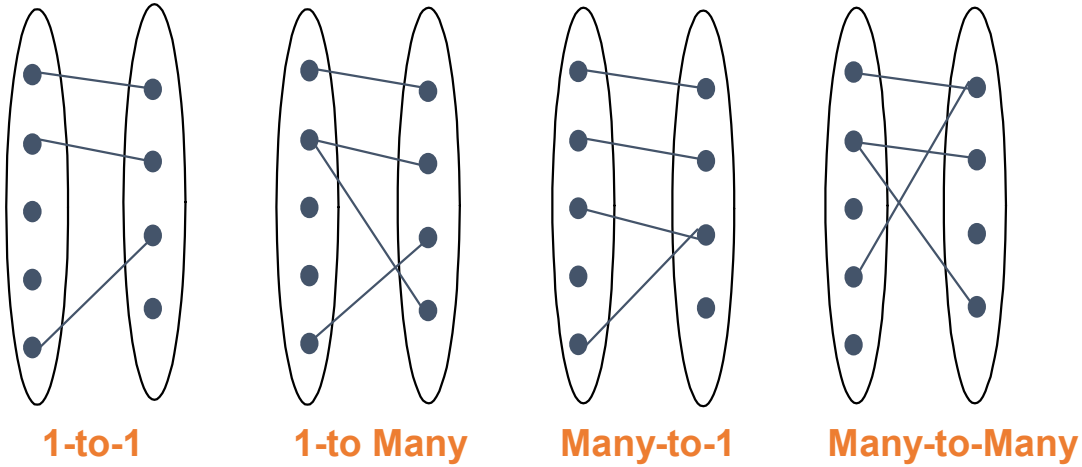

```
CREATE TABLE Employees
    (ssn VARCHAR(11),
    name VARCHAR(20),
    log  INTEGER,
    PRIMARY KEY  (ssn));
```

Review: participation and multiplicity Constraints

Lancaster University

- Each dept has at most one manager, according to the *key constraint* on Manages.



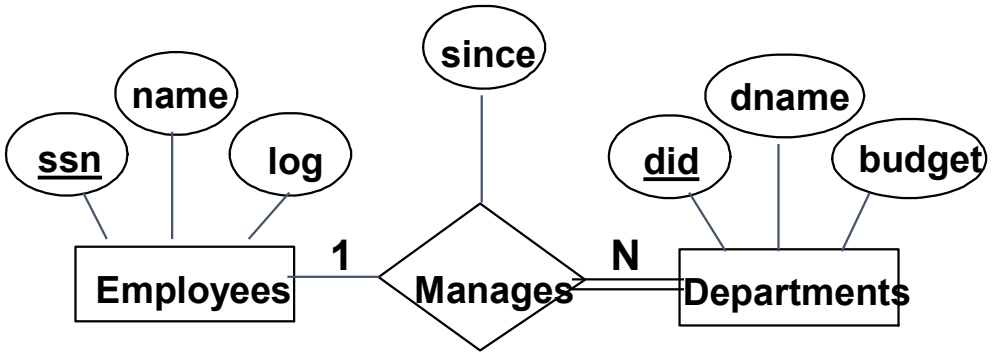


Translation to relational model?

Is this correct?

Lancaster University

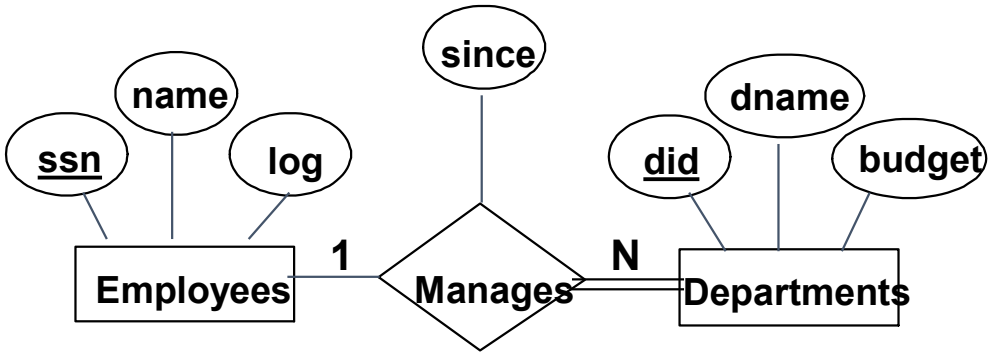
- “Employee can manage one department, a department must be managed by many employees”



Is this correct?

Lancaster University

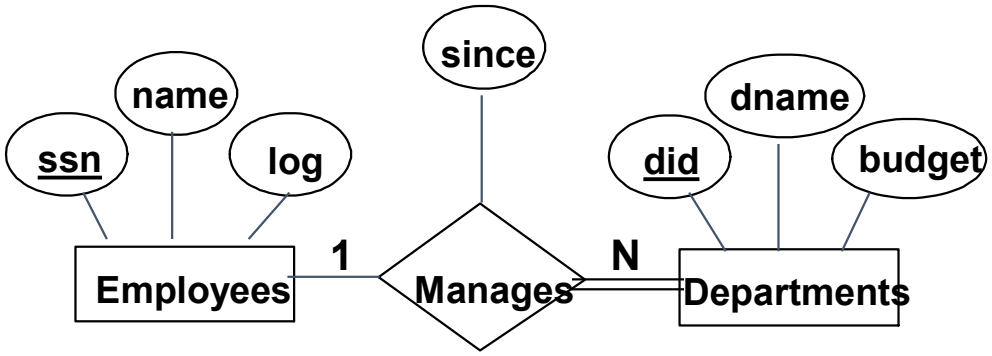
- “Employee must manage one department, a department may be managed by many employees”



Is this correct?

Lancaster University

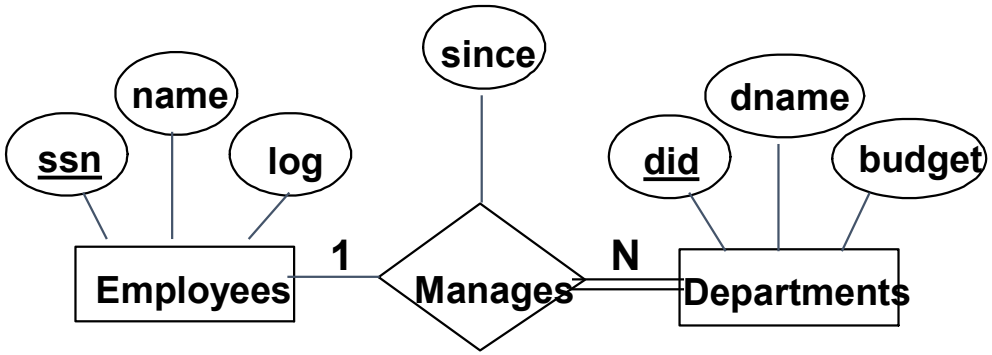
- “Employee can manage many departments, a department may be managed by many employees”



Is this correct?

Lancaster University

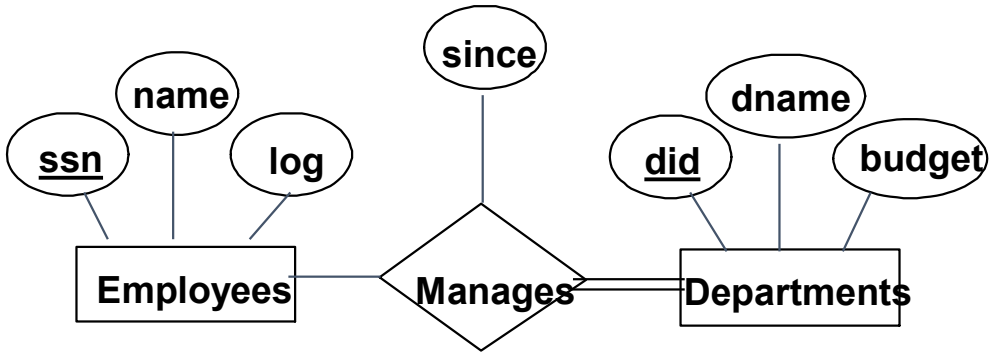
- “Employee can manage many departments, a department must be managed by one employee”



Is this correct?

Lancaster University

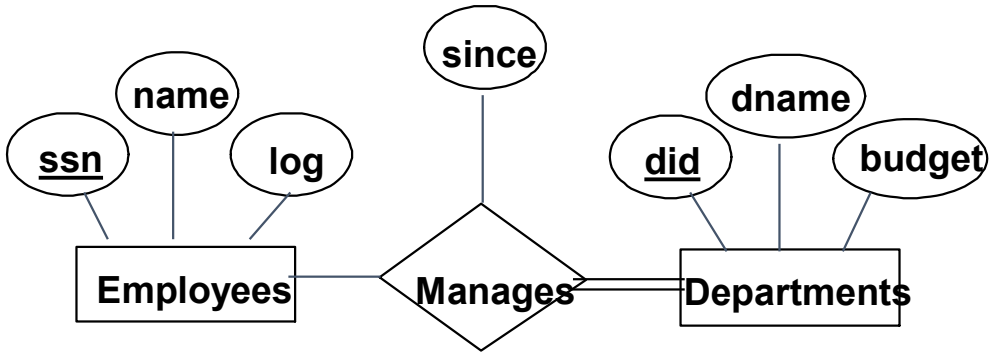
- “Employee must manage department(s) but department may be managed by employee(s).”



Is this correct?

Lancaster University

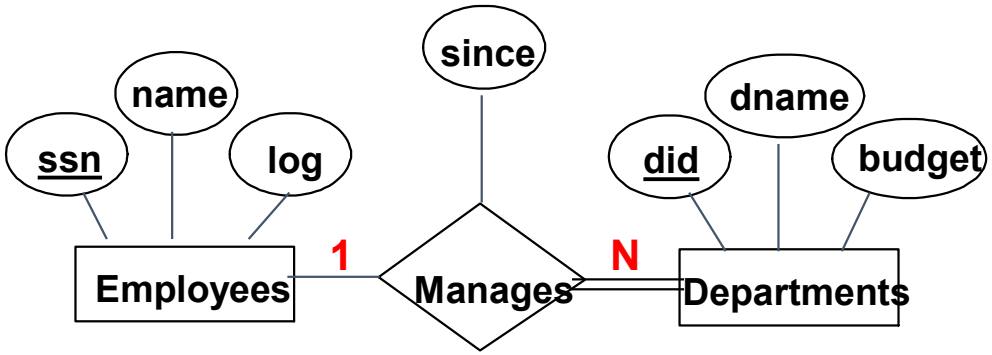
- “Employee can manage department(s) but department must be managed by employee(s).”



RECALL

Lancaster University

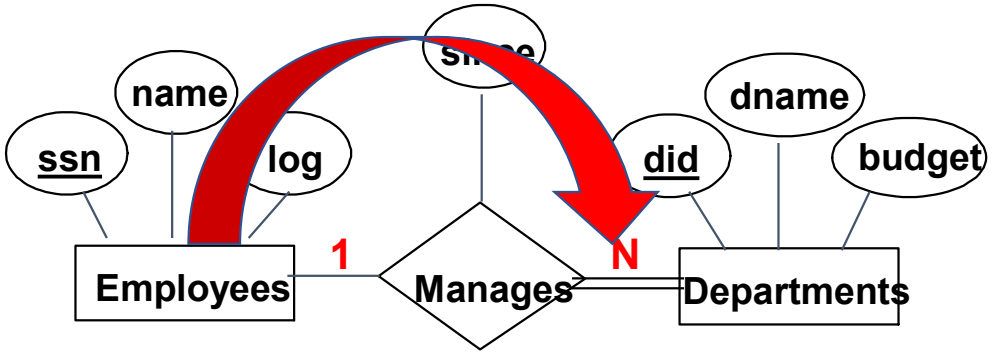
- Multiplicty constraints are given in the OPPOSITE ends of the relation



RECALL

Lancaster University

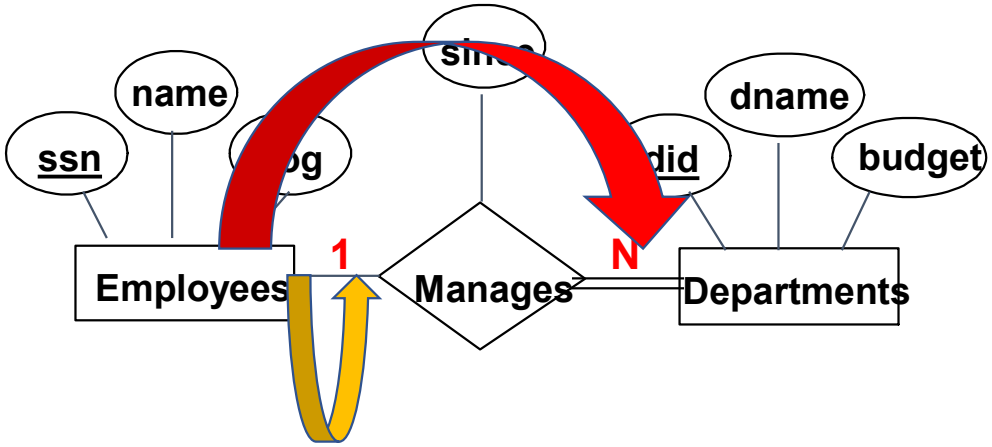
- Multiplicty constraints are given in the OPPOSITE ends of the relation
- Employee manages MANY departments.



RECALL

Lancaster University

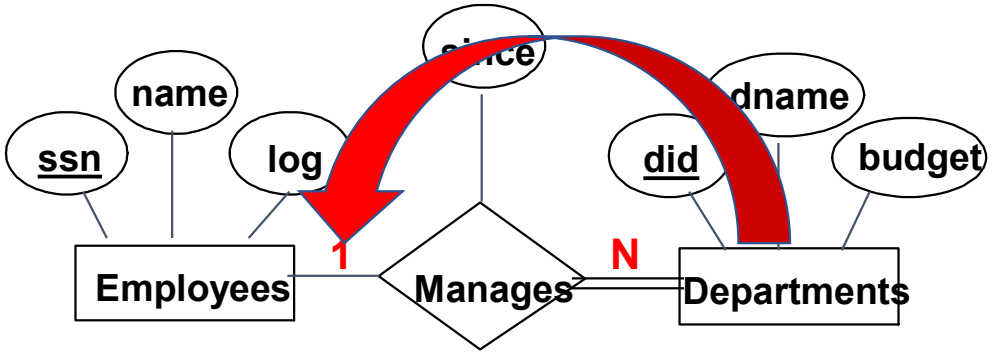
- Participation constraints are given BY the entity set.
- Employee MAY manage MANY departments.



RECALL

Lancaster University

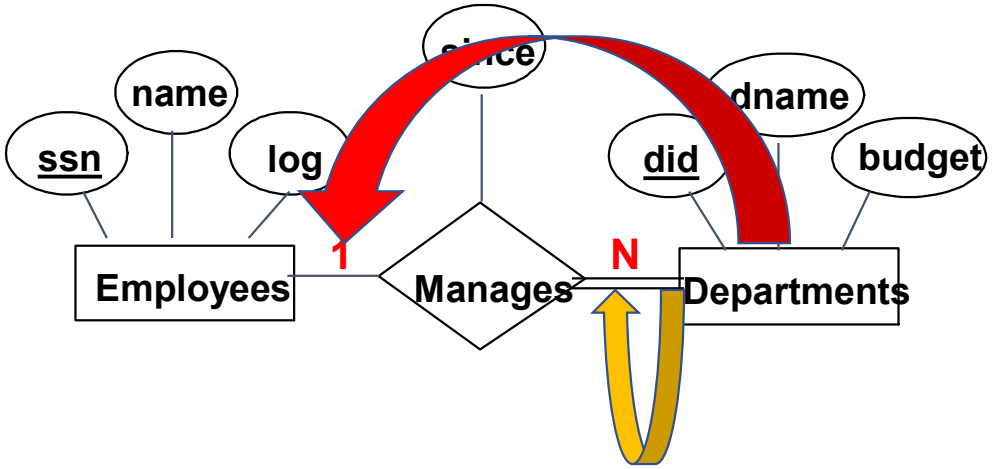
- Multiplicty constraints are given in the OPPOSITE ends of the relation, Participation constraints are given BY the entity set.
- Employee may manage many departments.
- Department managed by ONE employee.



RECALL

Lancaster University

- Multiplicty constraints are given in the OPPOSITE ends of the relation, Participation constraints are given BY the entity set.
- Employee may manage many departments.
- A department must be managed by one employee.



Foreign Keys in SQL

Lancaster University

FOREIGN KEY (*Attribute Name*) REFERENCES *Relation Name* (*Attribute Name*)

```
CREATE TABLE Enrolled
  (sid VARCHAR(20),  cid VARCHAR(20),  grade VARCHAR(2),
   PRIMARY KEY  (sid,cid),
   FOREIGN KEY (sid) REFERENCES Students (sid) );
```

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

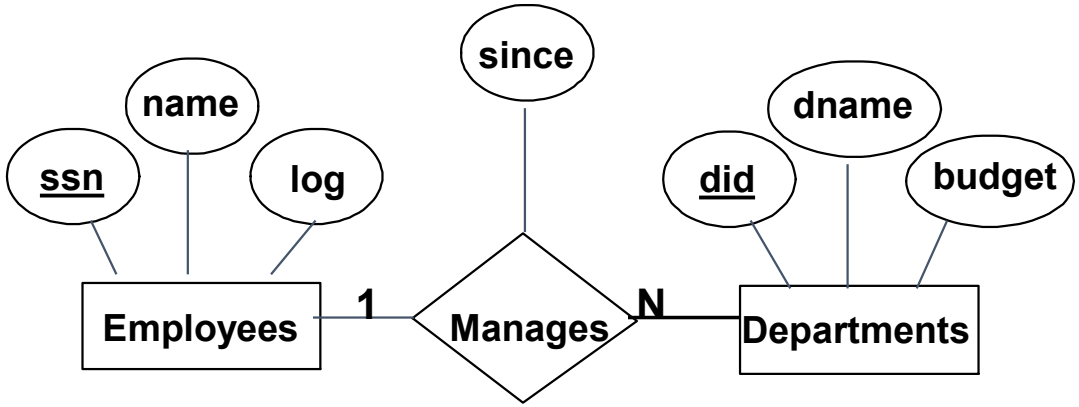
Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Review: Key Constraints

Lancaster University

- Each dept has at most one manager, according to the *key constraint* on Manages.



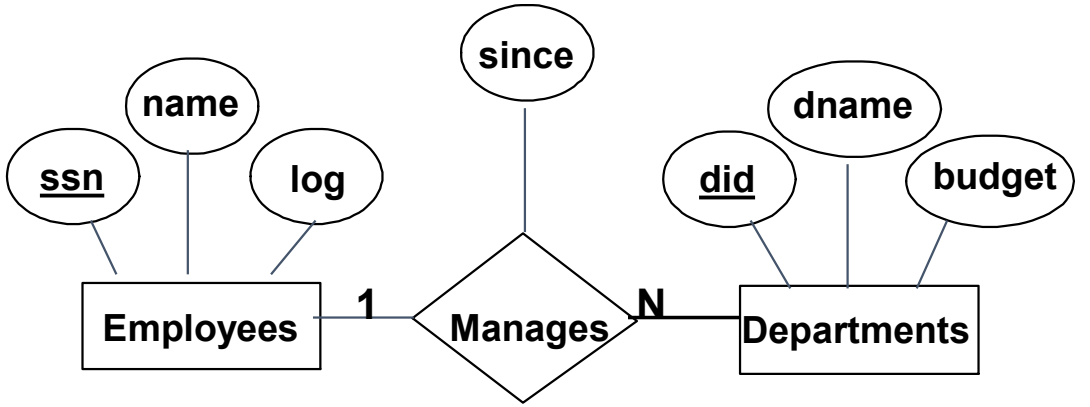

```
CREATE TABLE Employees
        (ssn VARCHAR(11),
        name VARCHAR(20),
        log  INTEGER,
        PRIMARY KEY  (ssn));

CREATE TABLE Departments
        (did INTEGER,
        dname VARCHAR(20),
        budget  REAL,
        PRIMARY KEY  (did));
```

Translation to relational model?

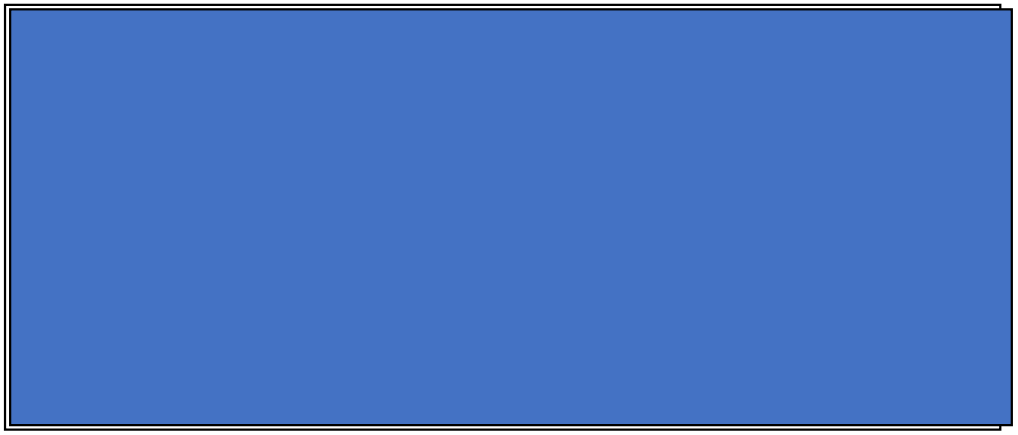
Review: Key Constraints

Lancaster University

- Each dept has at most one manager, according to the *key constraint* on Manages.



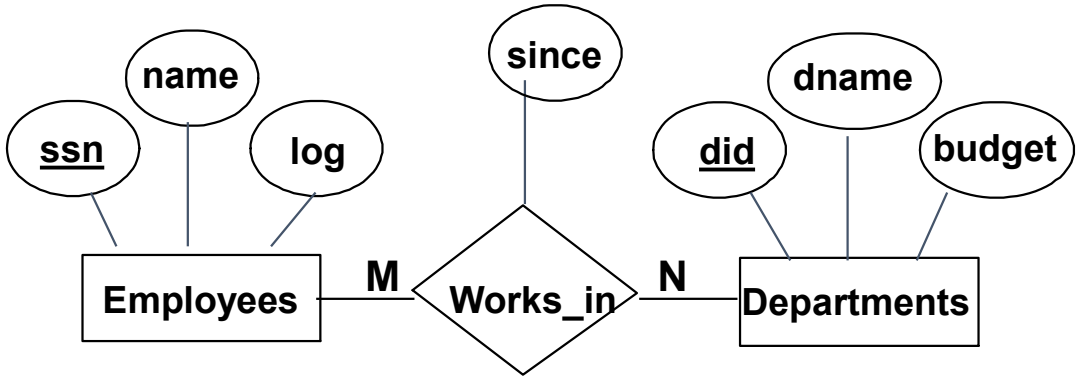
1. Since each department has a unique manager, we could instead combine Manages and Departments.



```
CREATE TABLE Employees
        (ssn VARCHAR(11),
        name VARCHAR(20),
        log  INTEGER,
        PRIMARY KEY  (ssn));
```

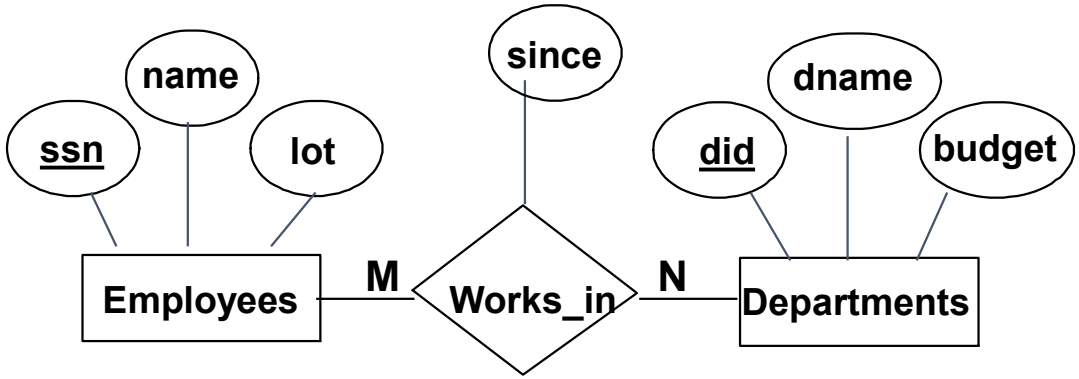
Relationship Sets to Tables

Lancaster University



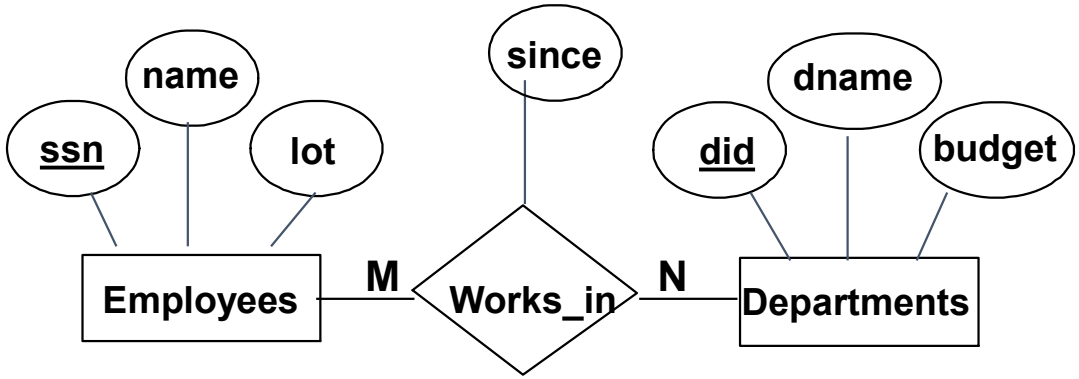
Relationship Sets to Tables

Lancaster University

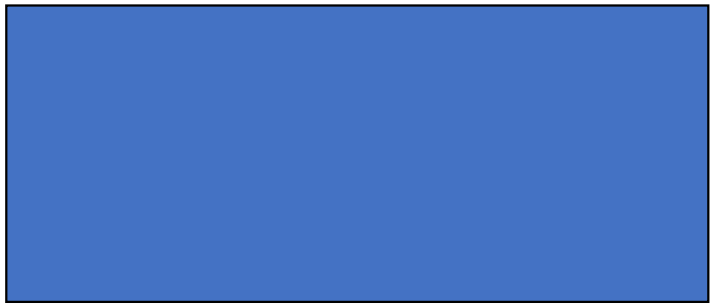


CREATE TABLE Works_In(

Relationship Sets to Tables

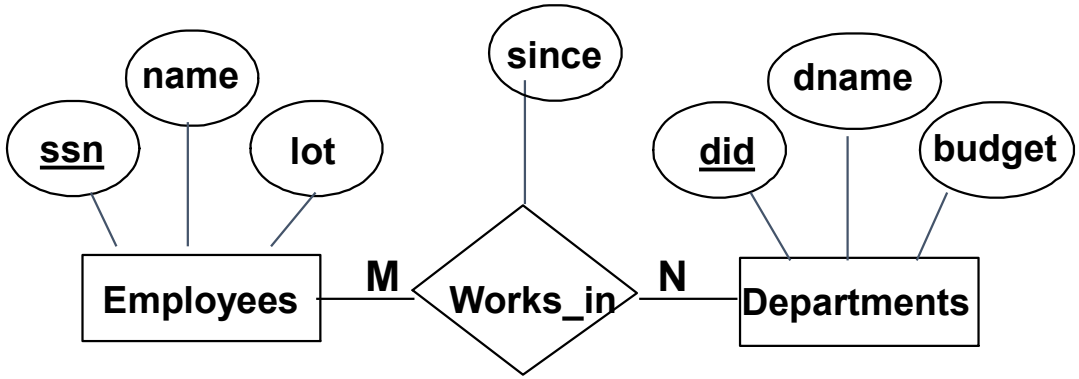



```
CREATE TABLE Works_In(
 ssn  VARCHAR(1),
  did  INTEGER,
  since  TEXT,
```



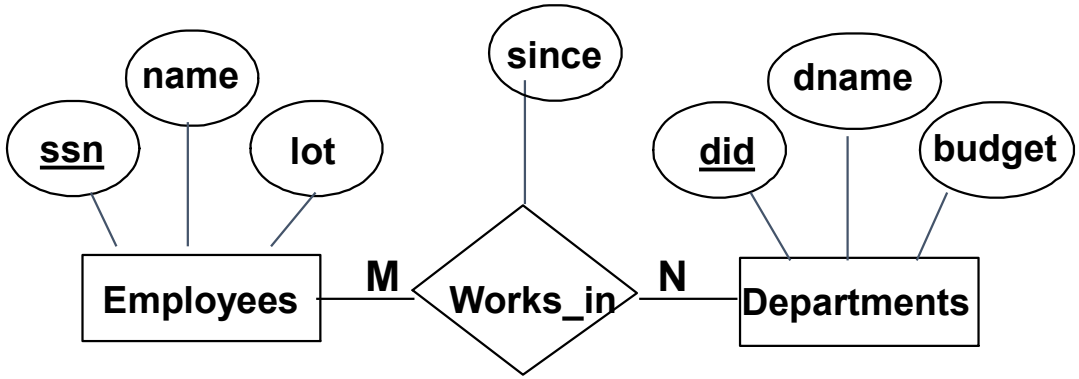
Relationship Sets to Tables

Lancaster University




```
CREATE TABLE Works_In(
 ssn  VARCHAR(1),
  did  INTEGER,
  since  TEXT,
  PRIMARY KEY (ssn, did),
```

Relationship Sets to Tables

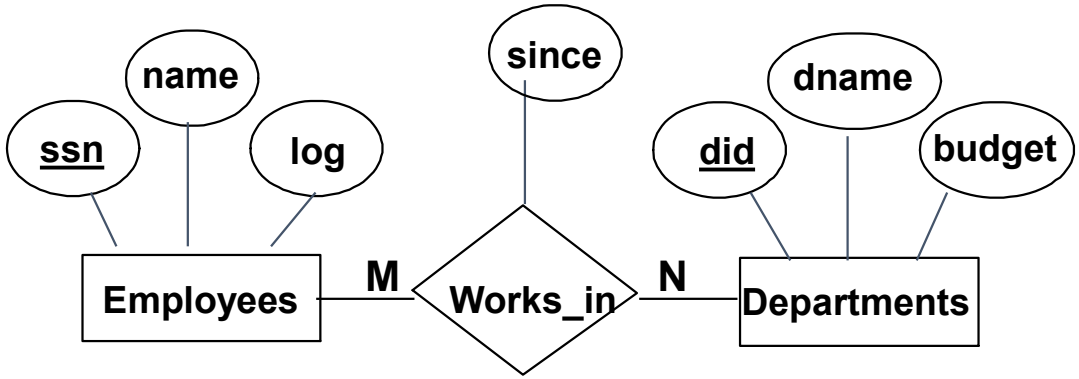



```
CREATE TABLE Works_In(
 ssn  VARCHAR(1),
  did  INTEGER,
  since  TEXT,
  PRIMARY KEY (ssn, did),
  FOREIGN KEY (ssn)
       REFERENCES Employees(ssn),
```

Relationship Sets to Tables

Lancaster University

- In translating a relationship set to a relation, attributes of the relation must include:
  - Keys for each participating entity set (as foreign keys).
  - All descriptive attributes.



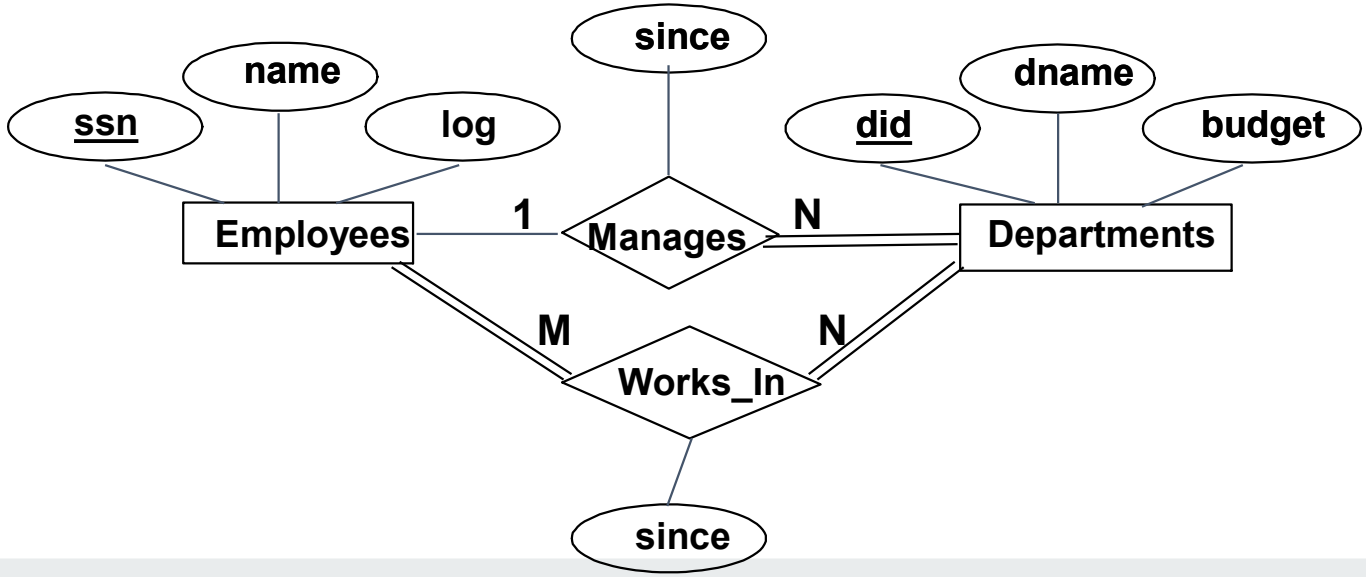

```
CREATE TABLE Employees
        (ssn VARCHAR(11),
        name VARCHAR(20),
        log  INTEGER,
        PRIMARY KEY  (ssn));

CREATE TABLE Departments
        (did INTEGER,
        dname VARCHAR(20),
        budget  REAL,
        PRIMARY KEY  (did));
```

```
CREATE TABLE Works_In(
 ssn  VARCHAR(1),
  did  INTEGER,
  since  TEXT,
  PRIMARY KEY (ssn, did),
FOREIGN KEY (ssn)
      REFERENCES Employees(ssn),
  FOREIGN KEY Departments (did)
       REFERENCES Departments (did) );
```

Review: Participation Constraints

Lancaster University

- Does every department have a manager?
  - If so, this is a *participation constraint*: the participation of Departments in Manages is said to be *total* (vs. *partial*).
    - Every *did* value in Departments table must appear in a row of the Manages table (with a non-null *ssn* value!)

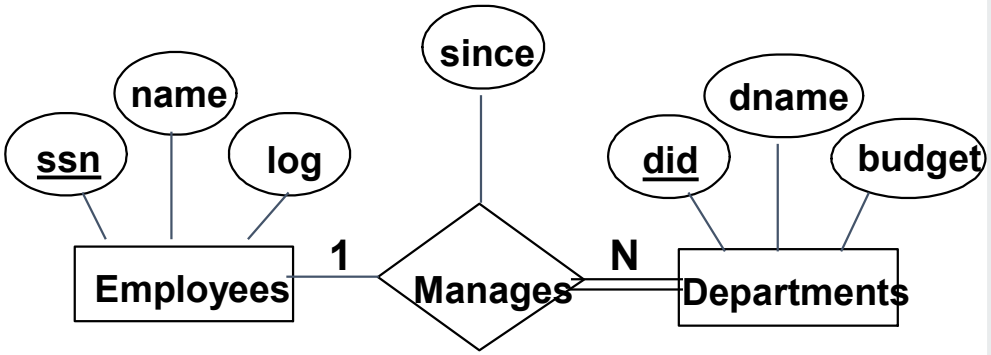


Participation Constraints in SQL

Lancaster University

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

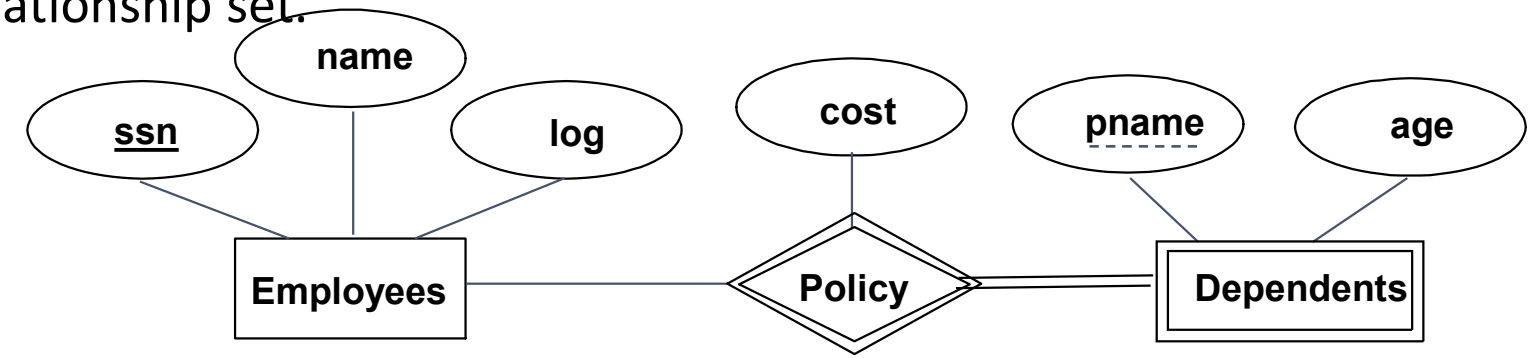
```
CREATE TABLE Dept_Mgr(
  did INTEGER,
  dname VARCHAR(20),
  budget REAL,
  since DATE,
  FOREIGN KEY (ssn) REFERENCES Employees(ssn)
ON DELETE CASCADE)
```



Review: Weak Entities

Lancaster University

- A *weak entity* can be identified uniquely only by considering the primary key of another (*owner*) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (1 owner, many weak entities).
  - Weak entity set must have total participation in this *identifying* relationship set.

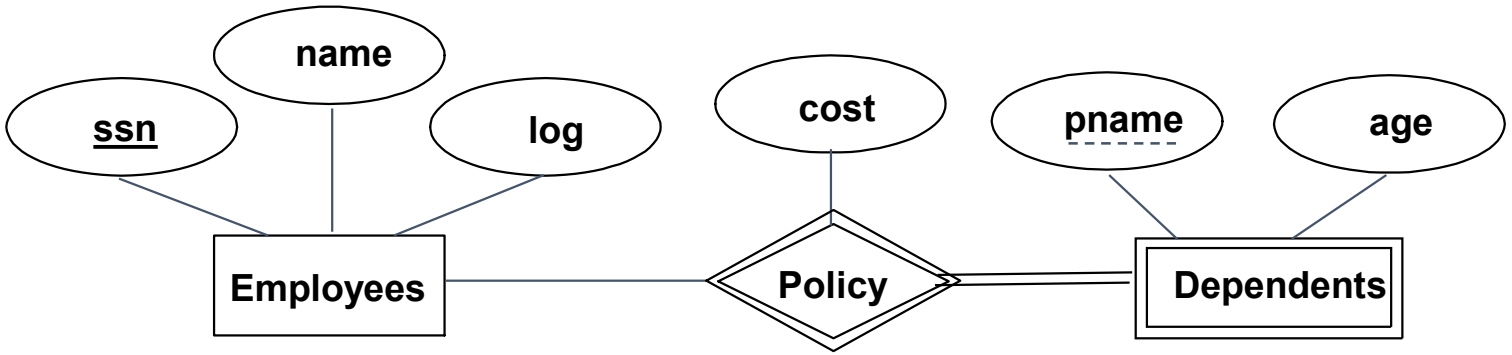


Weak Entities

Lancaster University

Weak entity set and identifying relationship set are translated into a single table.

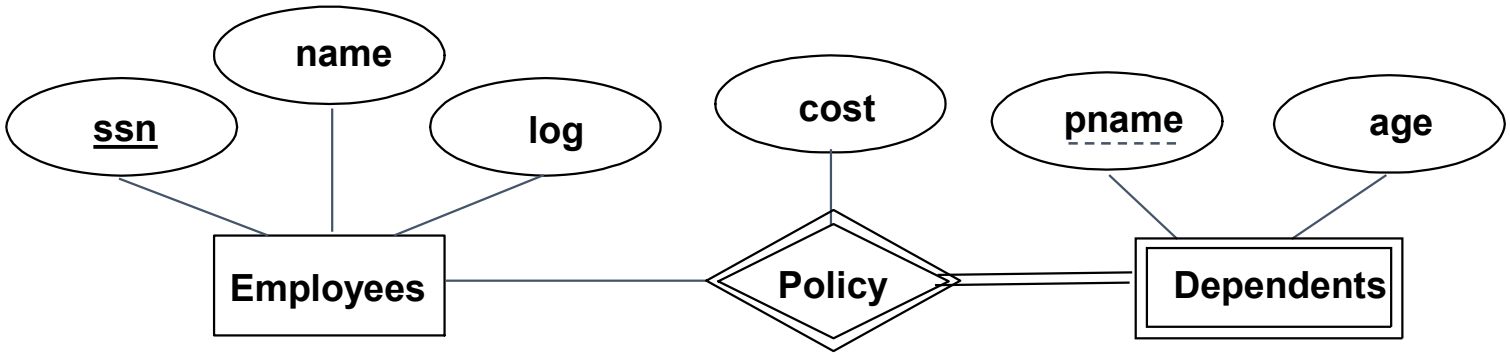
- When the owner entity is deleted, all owned weak entities must also be deleted.



Weak Entities

Lancaster University

```
CREATE TABLE  Dep_Policy (
  pname  VARCHAR(20),
  age  INTEGER,
  cost  REAL,
```



Referential Integrity in SQL/92

- SQL/92 supports all 4 options on deletes and updates.
  - Default is NO ACTION (*delete/update is rejected*)
  - CASCADE (also delete all tuples that refer to deleted tuple)
  - SET NULL / SET DEFAULT (sets foreign key value of referencing tuple)

DRAW ER DIAGRAM FOR THIS

```
CREATE TABLE Enrolled
  (sid VARCHAR(20),
   cid VARCHAR(20),
   grade VARCHAR(2),
   PRIMARY KEY (sid,cid),
   FOREIGN KEY (sid)
     REFERENCES Students (sid)
          ON DELETE CASCADE
          ON UPDATE SET DEFAULT );
```

EXAMPLE

Lancaster University

"For a given student and course, there is a single grade in Enrolled."

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Shero	shero@cs	18	3.2

```
CREATE TABLE Enrolled
  (sid VARCHAR(20)
   cid  VARCHAR(20),
   grade VARHAR(2),
   PRIMARY KEY  (sid,cid) );
```

UNIQUE keyword

Lancaster University

Create Enrolled table

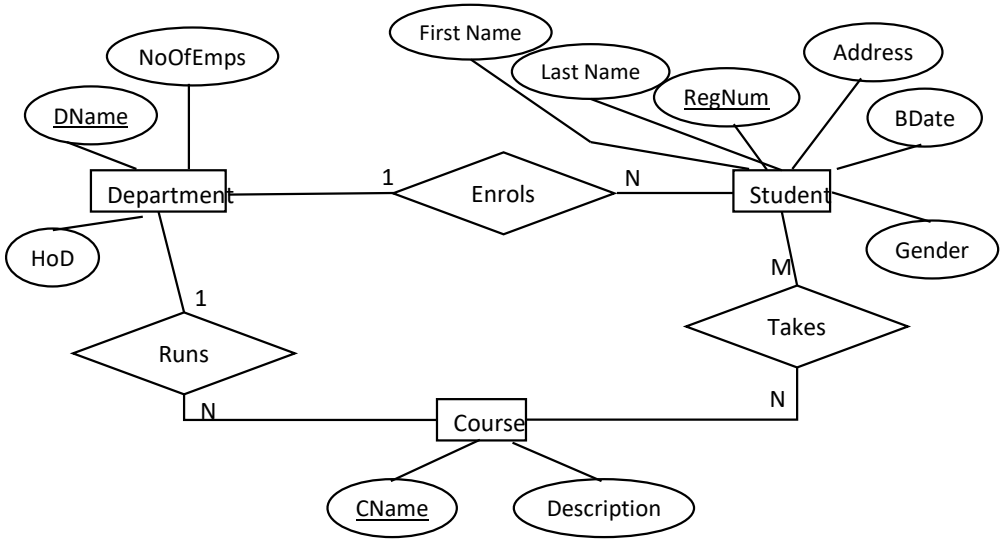
"**Students can take only one course, and receive a single grade for that course;** further, no two students in a course receive the same grade."

```
CREATE TABLE Enrolled
  (sid VARCHAR(20)
   cid  VARCHAR(20),
   grade VARCHAR(2),
   PRIMARY KEY  (sid),
   UNIQUE (cid, grade) );
```

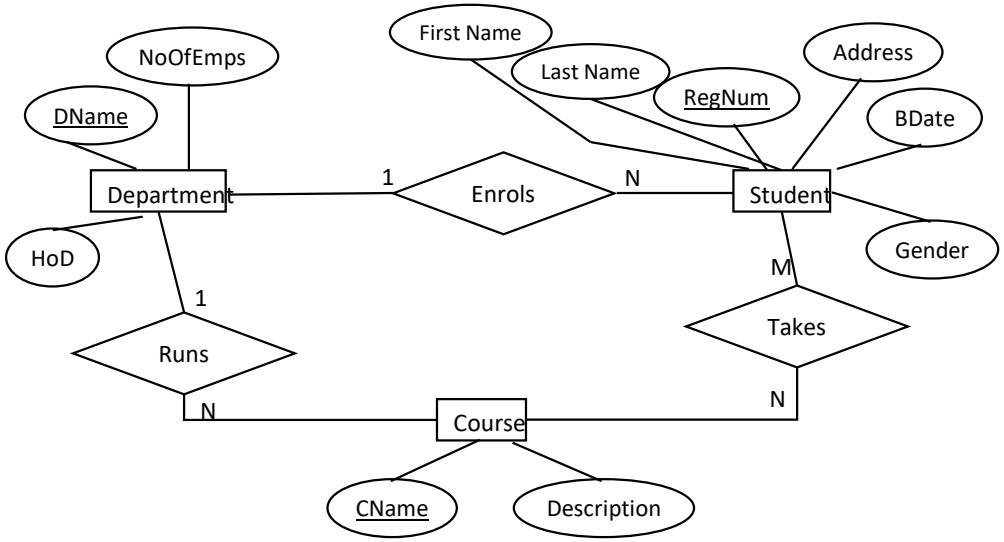
sid	cid	grade
53834	Carnatic101	C
53831	Reggae203	B
53650	Topology112	A
53666	History105	B

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Shero	shero@cs	18	3.2

Example

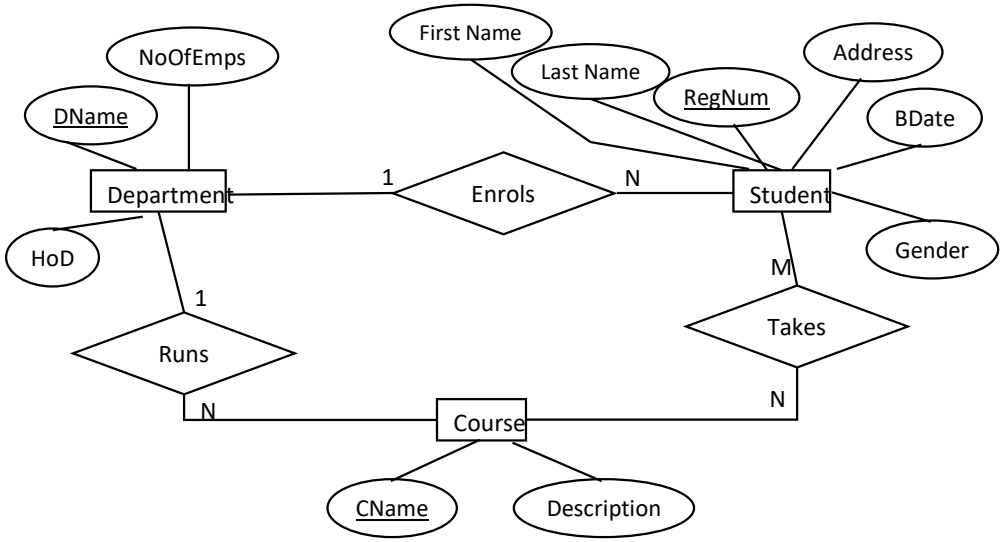

First Name
NoOfEmps
Last Name
Address
RegNum
DName
BDate
Department
1
Enrols
N
Student
HoD
Gender
M
1
Takes
Runs
N
N
Course
CName
Description

Example



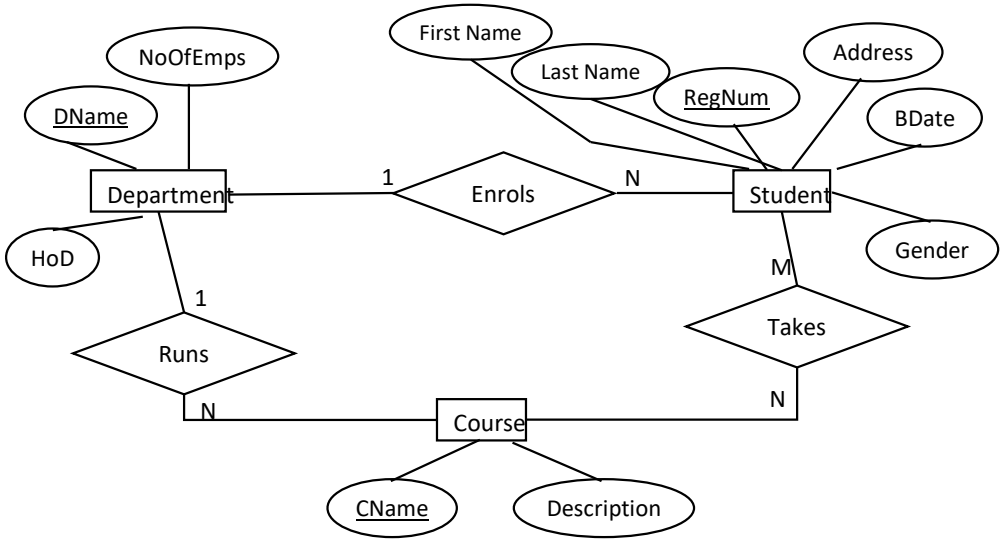
- CREATE TABLE Department(DName TEXT, HoD TEXT, NoOfEmp INTEGER, PRIMARY KEY(DName));

Example



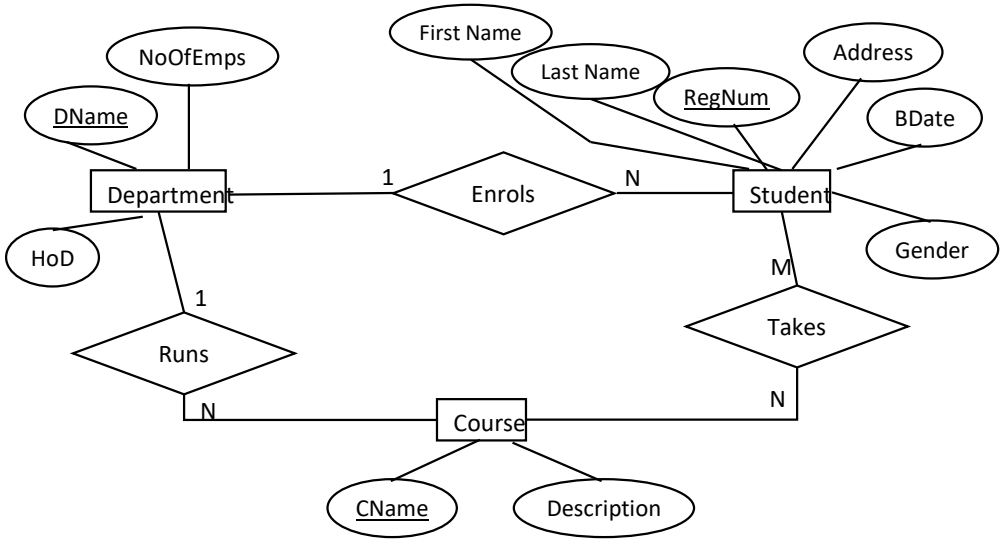
- CREATE TABLE Department(DName TEXT, HoD TEXT, NoOfEmp INTEGER, PRIMARY KEY(DName));
- CREATE TABLE StudentsEnrol(firstName TEXT, lastName TEXT, RegNumber INTEGER, Address TEXT, BDate TEXT, Gender TEXT, DepName TEXT, PRIMARY KEY (RegNumber), FOREIGN KEY(DepName) REFERENCES Department(DName) ON DELETE SET NULL);
-
-

Example



- CREATE TABLE Department(DName TEXT, HoD TEXT, NoOfEmp INTEGER, PRIMARY KEY(DName));
- CREATE TABLE StudentsEnrol(firstName TEXT, lastName TEXT, RegNumber INTEGER, Address TEXT, BDate TEXT, Gender TEXT, DepName TEXT, PRIMARY KEY (RegNumber), FOREIGN KEY(DepName) REFERENCES Department(DName) ON DELETE SET NULL);
- CREATE TABLE CourseRuns(CName TEXT, Description TEXT, DName TEXT, PRIMARY KEY (CName), FOREIGN KEY(DName) REFERENCES Department(DName) ON DELETE SET NULL);
-

Example



- CREATE TABLE Department(DName TEXT, HoD TEXT, NoOfEmp INTEGER, PRIMARY KEY(DName));
- CREATE TABLE StudentsEnrol(firstName TEXT, lastName TEXT, RegNumber INTEGER, Address TEXT, BDate TEXT, Gender TEXT, DepName TEXT, PRIMARY KEY (RegNumber), FOREIGN KEY(DepName) REFERENCES Department(DName) ON DELETE SET NULL);
- CREATE TABLE CourseRuns(CName TEXT, Description TEXT, DName TEXT, PRIMARY KEY (CName), FOREIGN KEY(DName) REFERENCES Department(DName) ON DELETE SET NULL);
- CREATE TABLE StTakesCourse(CName TEXT, RegNumber INTEGER, PRIMARY KEY(CName,RegNumber), FOREIGN KEY (CName) REFERENCES CourseRuns(CName), FOREIGN KEY (RegNumber) REFERENCES StudentsEnrol(RegNumber));

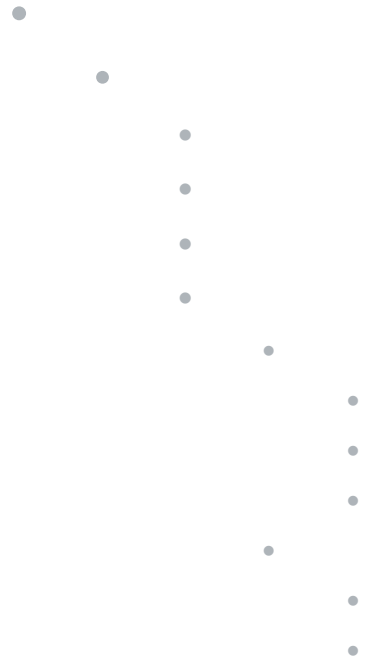
Lancaster University

Let’s begin ☺

Last lecture

Lancaster University

- We remembered



Last lecture

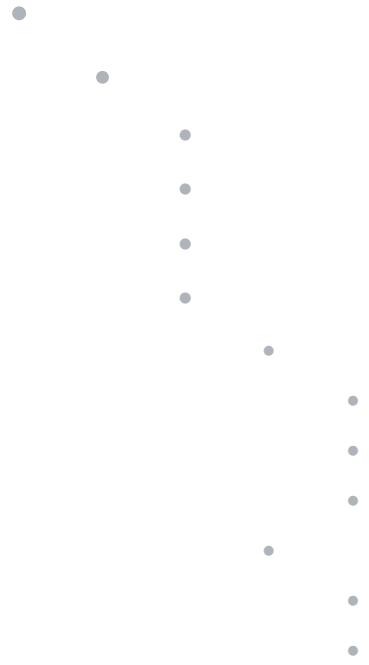
Lancaster University

- We remembered
  - Keys

Last lecture

Lancaster University

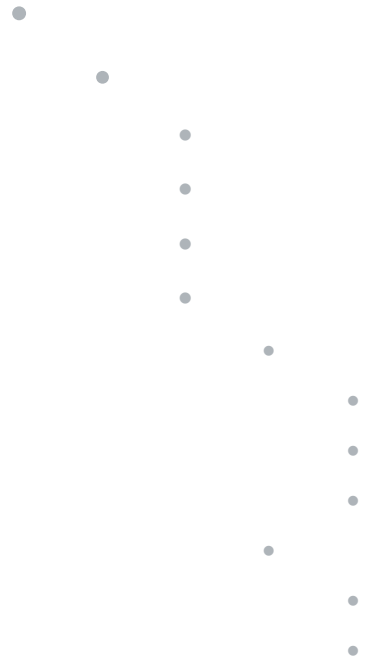
- We remembered
  - Keys
  - Relational Schema
  -



Last lecture

Lancaster University

- We remembered
  - Keys
  - Relational Schema
  - Participating and Multiplicity Constraints



Last lecture

Lancaster University

- We remembered
  - Keys
  - Relational Schema
  - Participating and Multiplicity Constraints
- We were introduced to:

Last lecture

Lancaster University

- We remembered
  - Keys
  - Relational Schema
  - Participating and Multiplicity Constraints

- We were introduced to:
  - Create Table
    - NOT NULL
    - Primary Key
    - UNIQUE
    -

Last lecture

Lancaster University

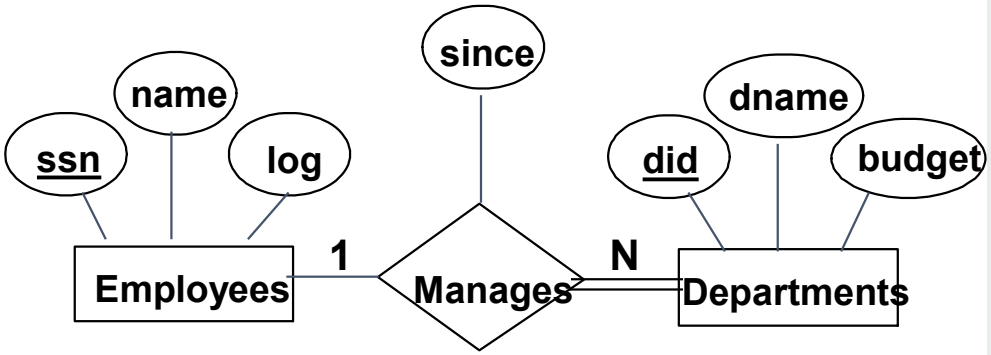
- We remembered
  - Keys
  - Relational Schema
  - Participating and Multiplicity Constraints
- We were introduced to:
  - Create Table
    - NOT NULL
    - Primary Key
    - UNIQUE
    - Foreign Key
      - ON DELETE
        - CASCADE
        - SET NULL
        - SET DEFAULT
      - ON UPDATE
        - SET DEFAULT
        - SET NULL

Integrity Constraints (Key,Participation) in SQL

Lancaster University

- We can capture participation constraints involving one entity set in a binary relationship, but little else (without resorting to CHECK constraints).

```
CREATE TABLE  Dept_Mgr(
  did  INTEGER,
  dname  VARCHAR(20),
  budget  REAL,
  ssn  VARCHAR(11) NOT NULL,
  since  DATE,
  PRIMARY KEY  (did),
  FOREIGN KEY  (ssn) REFERENCES Employees(ssn)
    ON DELETE CASCADE)
```



Some more introduction to SQL

Lancaster University

- Inserting and querying

Adding and Deleting Tuples

Lancaster University

One can insert a single tuple using

INSERT INTO <*Relation Name*> (<*Attributes*>) Values (<*all_the_values*>);

Directive

```
INSERT INTO Students (sid, name, login, age, gpa)
VALUES (53688, 'Smith', 'smith@ee', 18, 3.2);
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4

DELETE and SELECT

Lancaster University

The DELETE statement is used to delete existing records in a table.

```
DELETE
FROM Relation-set
WHERE <Conditions>;
```

Adding and Deleting Tuples

Lancaster University

Can delete all tuples satisfying some condition (e.g., name = Shero):

```
DELETE
FROM Students S
WHERE S.name = 'Jones';
```

Powerful variants of these commands are available; more later!

sid	name	login	age	gpa
53688	Smith	smith@ee	18	3.2

DELETE and SELECT

Lancaster University

The SELECT statement is used to select data from a database.
The data returned is stored in a result table, called the result set.

```
SELECT  Attributes
FROM  Relation-set
WHERE  <Conditions>;
```

If Attributes = * the query evaluates to $\sigma_{conditions}$

If Attributes = attr1, attr2… then the query evaluates to $\pi_{attr1,attr2....}(\sigma_{conditions})$

The SQL Query Language

If Attributes = * the query evaluates to $\sigma_{age>18}$

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Shero	shero@cs	18	3.2
53650	Shero	shero@math	19	3.8

Find all students with age 18

```
SELECT *
FROM Students S
WHERE S.age=18;
```

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Shero	shero@cs	18	3.2

The SQL Query Language

$$\pi_{name,login}(\sigma_{age>} \quad)$$

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Shero	shero@cs	18	3.2
53650	Shero	shero@math	19	3.8

Names and logins of all students with age 18

name	login
Jones	jones@cs
Shero	shero@cs

```
SELECT  S.name, S.login
FROM  Students S
WHERE  S.age=18;
```

Selection using Distinct to eliminate duplicates

- Basic query structure

```
SELECT      [DISTINCT]  attribute-list
FROM        relation-list
WHERE       condition
```

- DISTINCT is an optional keyword indicating that duplicates should be eliminated. (Otherwise duplicate elimination is not done)

```
mysql> select DISTINCT e.NAME FROM Enrolled2 e WHERE  NAME='URaz';
+------+
| NAME |
+------+
| URaz |
+------+
1 row in set (0.00 sec)
```

Selection using logical operators

```
SELECT  DISTINCT sname
FROM    Sailors
WHERE   Sailors.age>42 AND Sailors.rating>10
```

```
SELECT      [DISTINCT]  attribute-list
FROM        relation-list
WHERE       condition
```

- Conditions (ATTR *op* CONST or ATTR1 *op* ATTR2, where *op* is one of ($<, >, =, \leq, \geq, \neq$) combined using AND, OR and NOT.

These are logical operations!

What will happen if more than relations are used: Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  -
  -
  -
  -
-

```
SELECT      [DISTINCT] attribute-list
FROM        relation-list
WHERE       condition
```

What will happen if more than relations are used: Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  -
  -
  -
-

```
SELECT      [DISTINCT]  attribute-list
FROM        relation-list
WHERE       condition
```

What will happen if more than relations are used: Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they do not satisfy the *conditions*.
  -
  -
-

```
SELECT      [DISTINCT]  attribute-list
FROM        relation-list
WHERE       condition
```

What will happen if more than relations are used: Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they do not satisfy the *conditions*.
  - Display attributes that are in *attribute-list*.
  -

```
SELECT      [DISTINCT]  attribute-list
FROM        relation-list
WHERE       condition
```

What will happen if more than relations are used: Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they do not satisfy the *conditions*.
  - Display attributes that are in *attribute-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
-

```
SELECT      [DISTINCT]  attribute-list
FROM        relation-list
WHERE       condition
```

What will happen if more than relations are used: Conceptual Evaluation Strategy

- Semantics of an SQL query defined in terms of the following conceptual evaluation strategy:
  - Compute the cross-product of *relation-list*.
  - Discard resulting tuples if they do not satisfy the *conditions*.
  - Display attributes that are in *attribute-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
- This strategy is probably the least efficient way to compute a query! An optimiser will find more efficient strategies to compute *the same answers*.

```
SELECT      [DISTINCT] attribute-list
FROM        relation-list
WHERE       condition
```

Example of Conceptual Evaluation

```
SELECT    [DISTINCT] attribute-list
FROM      relation-list
WHERE     condition
```

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid AND Reserves.bid=103
```

$$\pi_{sname}(Sailors \bowtie_{Sailors.sid=Reserves.sid \wedge age>18} Reserves)$$

Query: Find names of sailors who Reserved boat number 103

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

Range Variables

Query: Find names
of sailors who
Reserved boat
number 103

```
SELECT  sname
FROM    Sailors, Reserves
WHERE   Sailors.sid=Reserves.sid AND Reserves.bid=103
```

```
SELECT  S.sname
FROM    Sailors S, Reserves R
WHERE   S.sid=R.sid AND R.bid=103
```

Range variables are necessary when joining a table with itself !!!

Expressions and Strings

```
SELECT    [DISTINCT] attribute-list
FROM      relation-list
WHERE     condition
```

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: *Find triples (of ages of sailors and two new attributes defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
-
-

Expressions and Strings

```
SELECT      [DISTINCT] attribute-list
FROM        relation-list
WHERE       condition
```

```
SELECT  S.age, age1=S.age-5, 2*S.age AS age2
FROM  Sailors S
WHERE  S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: *Find triples (of ages of sailors and two new attributes defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
- **AS and = are two ways to name fields in result.**
-

Expressions and Strings

```
SELECT    [DISTINCT] attribute-list
FROM      relation-list
WHERE     condition
```

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: *Find triples (of ages of sailors and two new attributes defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
- **AS and = are two ways to name fields in result.**
- **LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.**

Expressions and Strings

```
SELECT    [DISTINCT] attribute-list
FROM      relation-list
WHERE     condition
```

```
SELECT S.age, age1=S.age-5, 2*S.age AS age2
FROM Sailors S
WHERE S.sname LIKE 'B_%B'
```

- Illustrates use of arithmetic expressions and string pattern matching: *Find triples (of ages of sailors and two new attributes defined by expressions) for sailors whose names begin and end with B and contain at least three characters.*
- **AS and = are two ways to name fields in result.**
- **LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.**

Note: we can create attributes! In this example they are age1 and age2

Find sid's of sailors who've reserved a red or a green boat

B

BID	Colr
b1	red
b2	grn

R

SID	BID
s1	b1
s1	b2
s2	b1

- UNION: Can be used to compute the union of any two *union-compatible* sets of tuples (which are themselves the result of SQL queries).

```
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid
  AND (B.color='red' OR B.color='green')
```

```
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid
          AND B.color='red'
UNION
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid
          AND B.color='green'
```

B.BID	B.Colr	R.SID	R.BID
b1	red	s1	b1
b1	red	s1	b2
b1	red	s2	b1
b2	grn	s1	b1
b2	grn	s1	b2
b2	grn	s2	b1

Find sid's of sailors who've reserved a red but not a green boat

- EXCEPT : Used to compute the set difference of two *union-compatible* sets of tuples
- What do we get if we replace **UNION** with **EXCEPT** in the previous SQL query?

```
SELECT  R.sid
FROM  Boats B, Reserves R        A
WHERE  R.bid=B.bid
          AND B.color='red'      -
EXCEPT
SELECT  R.sid                    B
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid
          AND B.color='green'
```

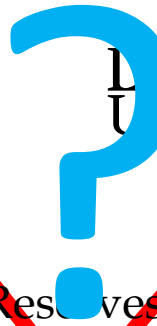
B

BID	Colr
b1	red
b2	grn

R

SID	BID
s1	b1
s1	b2
s2	b1

Find sid's of sailors who've reserved a red and a green boat




```
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid
  AND (B.color='red' AND B.color='green')
```

- AND : Used to compute the set intersection of two *union-compatible* sets of tuples

B

BID	Colr
b1	red
b2	grn

R

SID	BID
s1	b1
s1	b2
s2	b1

B1 X R1

B.BID	B.Colr	R.SID	R.BID
b1	red	s1	b1
b1	red	s1	b2
b1	red	s2	b1
b2	grn	s1	b1
b2	grn	s1	b2
b2	grn	s2	b1

Find sid's of sailors who've reserved a red and a green boat

Lancaster University

B1 X R1

BID	Colr	SID	BID
b1	red	s1	b1
b1	red	s1	b2
b1	red	s2	b1
b2	grn	s1	b1
b2	grn	s1	b2
b2	grn	s2	b1

Let us use *Tmp*1 to for sid's of sailors who reserved red boats

B2 X R2

BID	Colr	SID	BID
b1	red	s1	b1
b1	red	s1	b2
b1	red	s2	b1
b2	grn	s1	b1
b2	grn	s1	b2
b2	grn	s2	b1

Let us use *Tmp*2 to for sid's of sailors who reserved green boats

Find sid's of sailors who've reserved a red and a green boat

Lancaster University

B1 X R1

BID	Colr	SID	BID
b1	red	s1	b1
b1	red	s1	b2
b1	red	s2	b1
b2	grn	s1	b1
b2	grn	s1	b2
b2	grn	s2	b1

Let us use *Tmp*1 to for sid's of sailors who reserved red boats

Take cross product of these two relations.

B2 X R2

BID	Colr	SID	BID
b1	red	s1	b1
b1	red	s1	b2
b1	red	s2	b1
b2	grn	s1	b1
b2	grn	s1	b2
b2	grn	s2	b1

Let us use *Tmp*2 to for sid's of sailors who reserved green boats

B2X R2 X B1 X R1

BID	Colr	SID	BID	BID	Colr	SID	BID
b1	red	s1	b1	b1	red	s1	b1
b1	red	s1	b1	b1	red	s1	b2
b1	red	s1	b1	b1	red	s2	b1
b1	red	s1	b1	b2	grn	s1	b1
b1	red	s1	b1	b2	grn	s1	b2
b1	red	s1	b1	b2	grn	s2	b1
b1	red	s1	b2	b1	red	s1	b1
b1	red	s1	b2	b1	red	s1	b2
b1	red	s1	b2	b1	red	s2	b1
b1	red	s1	b2	b2	grn	s1	b1
b1	red	s1	b2	b2	grn	s1	b2
b1	red	s1	b2	b2	grn	s2	b1
b1	red	s2	b1	b1	red	s1	b1
b1	red	s2	b1	b1	red	s1	b2
b1	red	s2	b1	b1	red	s2	b1
b1	red	s2	b1	b2	grn	s1	b1
b1	red	s2	b1	b2	grn	s1	b2
b1	red	s2	b1	b2	grn	s2	b1

Lancaster University

B2X R2 X B1 X R1 (Continue)

BID	Colr	SID	BID	BID	Colr	SID	BID
b1	grn	s1	b1	b1	red	s1	b1
b1	grn	s1	b1	b1	red	s1	b2
b1	grn	S1	b1	b1	red	s2	b1
b1	grn	s1	b1	b2	grn	s1	b1
b1	grn	s1	b1	b2	grn	s1	b2
b1	grn	s1	b1	b2	grn	s2	b1
b1	grn	s1	b2	b1	red	s1	b1
b1	grn	s1	b2	b1	red	s1	b2
b1	grn	s1	b2	b1	red	s2	b1
b1	grn	s1	b2	b2	grn	s1	b1
b1	grn	s1	b2	b2	grn	s1	b2
b1	grn	s1	b2	b2	grn	s2	b1
b1	grn	s2	b1	b1	red	s1	b1
b1	grn	s2	b1	b1	red	s1	b2
b1	grn	s2	b1	b1	red	s2	b1
b1	grn	s2	b1	b2	grn	s1	b1
b1	grn	s2	b1	b2	grn	s1	b2
b1	grn	s2	b1	b2	grn	s2	b1

Lancaster University

```
SELECT  R1.sid
FROM  Boats B1, Reserves R1, Boats B2,
Reserves R2
WHERE  R1.sid = R2.sid AND  R1.bid=B1.bid
AND R2.bid=B2.bid
  AND (B1.color='red'  AND B2.color='green')
```

Find sid's of sailors who've reserved a red and a green boat

B2X R2 X B1 X R1 (Continue)

BID	Colr	SID	BID	BID	Colr	SID	BID
b1	***grn***	***s1***	***b1***	***b1***	***red***	***s1***	***b1***
b1	***grn***	***s1***	***b1***	***b1***	***red***	***s1***	***b2***
b1	***grn***	***S1***	***b1***	***b1***	***red***	***s2***	***b1***
b1	grn	s1	b1	b2	grn	s1	b1
b1	grn	s1	b1	b2	grn	s1	b2
b1	grn	s1	b1	b2	grn	s2	b1
b1	***grn***	***s1***	***b2***	***b1***	***red***	***s1***	***b1***
b1	***grn***	***s1***	***b2***	***b1***	***red***	***s1***	***b2***
b1	***grn***	***s1***	***b2***	***b1***	***red***	***s2***	***b1***
b1	grn	s1	b2	b2	grn	s1	b1
b1	grn	s1	b2	b2	grn	s1	b2
b1	grn	s1	b2	b2	grn	s2	b1
b1	***grn***	***s2***	***b1***	***b1***	***red***	***s1***	***b1***
b1	***grn***	***s2***	***b1***	***b1***	***red***	***s1***	***b2***
b1	***grn***	***s2***	***b1***	***b1***	***red***	***s2***	***b1***
b1	grn	s2	b1	b2	grn	s1	b1
b1	grn	s2	b1	b2	grn	s1	b2
b1	grn	s2	b1	b2	grn	s2	b1

SELECT R1.sid
FROM Boats B1, Reserves R1, Boats B2, Reserves R2
WHERE R1.sid = R2.sid AND R1.bid=B1.bid
AND R2.bid=B2.bid
AND (***B1.color='red' AND B2.color='green'***)

Lancaster University

B2X R2 X B1 X R1 (Continue)

BID	Colr	SID	BID	BID	Colr	SID	BID
b1	***grn***	***s1***	***b1***	***b1***	***red***	***s1***	***b1***
b1	***grn***	***s1***	***b1***	***b1***	***red***	***s1***	***b2***
b1	***grn***	***S1***	***b1***	***b1***	***red***	***s2***	***b1***
b1	grn	s1	b2	b1	red	s1	b1
b1	grn	s1	b2	b1	red	s1	b2
b1	grn	s1	b2	b1	red	s2	b1
b1	***grn***	***s2***	***b1***	***b1***	***red***	***s1***	***b1***
b1	***grn***	***s2***	***b1***	***b1***	***red***	***s1***	***b2***
b1	***grn***	***s2***	***b1***	***b1***	***red***	***s2***	***b1***

SELECT R1.sid
FROM Boats B1, Reserves R1, Boats B2, Reserves R2
WHERE R1.sid = R2.sid AND R1.bid=B1.bid
AND ***R2.bid=B2.bid***
AND (B1.color='red' AND B2.color='green')

Lancaster University

B2X R2 X B1 X R1 (Continue)

BID	Colr	SID	BID	BID	Colr	SID	BID
b1	***grn***	***s1***	***b1***	***b1***	***red***	***s1***	***b1***
b1	grn	s1	b1	b1	red	s1	b2
b1	***grn***	***S1***	***b1***	***b1***	***red***	***s2***	***b1***
b1	***grn***	***s2***	***b1***	***b1***	***red***	***s1***	***b1***
b1	grn	s2	b1	b1	red	s1	b2
b1	***grn***	***s2***	***b1***	***b1***	***red***	***s2***	***b1***

```
SELECT  R1.sid
FROM  Boats B1, Reserves R1, Boats B2,
Reserves R2
WHERE  R1.sid = R2.sid AND  R1.bid=B1.bid
AND R2.bid=B2.bid
  AND (B1.color='red' AND B2.color='green')
```

Lancaster University

B2X R2 X B1 X R1 (Continue)

BID	Colr	SID	BID	BID	Colr	SID	BID
b1	***grn***	***s1***	***b1***	***b1***	***red***	***s1***	***b1***
b1	grn	S1	b1	b1	red	s2	b1
b1	grn	s2	b1	b1	red	s1	b1
b1	grn	s2	b1	b1	red	s2	b1

Lancaster University

SELECT R1.sid
FROM Boats B1, Reserves R1, Boats B2, Reserves R2
WHERE ***R1.sid = R2.sid*** AND R1.bid=B1.bid
AND R2.bid=B2.bid
AND (B1.color='red' AND B2.color='green')

B2X R2 X B1 X R1 (Continue)

BID	Colr	SID	BID	BID	Colr	SID	BID
b1	***grn***	***s1***	***b1***	***b1***	***red***	***s1***	***b1***

SO the answer to the query is s1

Lancaster University

Find names of sailors who've reserved a red and a green boat.

sid	sname	rating	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	bid	day
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	bname	color
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

$$\rho(Tempred, \pi_{sid}((\sigma_{color='red'} Boats) \bowtie Reserves))$$

$$\rho\ (Tempgreen, \pi_{sid}((\sigma_{color='green'} Boats) \bowtie Reserves))$$

$$\pi_{sname}((Tempred \cap Tempgreen) \bowtie Sailors)$$

Find sid's of sailors who've reserved a red and a green boat

Lancaster University

- INTERSECT: Can be used to compute the intersection of any two *union-compatible* sets of tuples.
- Included in the SQL/92 standard, but some systems don't support it.
- Contrast symmetry of the UNION and INTERSECT queries with how much the other versions differ.

```
SELECT  R.sid
FROM  Boats B1, Reserves R1,
          Boats B2, Reserves R2
WHERE  R1.sid = R2.sid AND
     R1.bid=B1.bid AND R2.bid=B2.bid
  AND (B1.color='red' AND B2.color='green')
```

```
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE  R.bid=B.bid
            AND B.color='red'
INTERSECT
SELECT  R.sid
FROM  Boats B, Reserves R
WHERE   R.bid=B.bid
            AND B.color='green'
```

Nested Queries

Lancaster University

Find names of sailors who've reserved boat #103:

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid  (ONLY ONE Colum)
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

- IN returns true if the first value is contained in the set returned (single column!
-)WHERE clause can itself contain an SQL query! (As well as FROM and HAVING clauses which we will see later on.)

Nested Queries

```
SELECT  S.sname
FROM  Sailors S
WHERE  S.sid IN  (SELECT  R.sid
                  FROM  Reserves R
                  WHERE  R.bid=103)
```

Lancaster University

Find names of sailors who've reserved boat #103:

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Nested Queries

Lancaster University

Find names of sailors
who did NOT reserve boat #103:

```
SELECT S.sname
FROM Sailors S
WHERE S.sid NOT IN (SELECT R.sid
                    FROM Reserves R
                    WHERE R.bid=103)
```

Find names of sailors
Who reserved a boat that is not boat #103:
Answer must be 64

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Nested Queries with Correlation

Lancaster University

Find names of sailors who've reserved boat #103:

- EXISTS returns true TRUE if the set, is nonempty. EXISTS operator is another set comparison operator, like IN.

```
SELECT S.sname
FROM Sailors S
WHERE EXISTS (SELECT *
              FROM Reserves R
              WHERE R.bid=103 AND S.sid=R.sid)
```

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

- To understand semantics of correlated queries, think of a *nested loops* evaluation: *For each Sailors tuple, check the qualification by computing the subquery.*

```
For (i=1…10) do  {
        For (j=1…5)  do {
            y=i+1;
        }
}
```

Nested Queries with Correlation

Lancaster University

Find names of sailors who reserved a boat at most once

- UNIQUE construct can be used.
- UNIQUE checks for duplicate tuples. Returns TRUE if the corresponding set **does not** contain duplicates.

```
SELECT  S.sname
FROM  Sailors S
WHERE  UNIQUE (SELECT  R.bid
                FROM  Reserves R
                WHERE  S.sid=R.sid)
```

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

More on Set-Comparison Operators

Lancaster University

- We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *condition* ANY, condition ALL $>, <, =, \geq, \leq, \neq$ '<>'
- ***Find sailors whose rating is greater than that of some sailor called Horatio:***

ANY (returns true if there exist tuples (returned from nested WHERE clause) obey the condition!)
ALL (returns true if all tuples (returned from nested WHERE clause) obey the condition!)

```
SELECT *
FROM Sailors S
WHERE S.rating > ANY (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.sname='Horatio')
```

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

More on Set-Comparison Operators

Lancaster University

- We've already seen IN, EXISTS and UNIQUE. Can also use NOT IN, NOT EXISTS and NOT UNIQUE.
- Also available: *condition* ANY, condition ALL $>, <, =, \geq, \leq, \neq$ '<>'
- ***Find sailors whose rating is greater than that of ALL sailor's called Horatio:***

ANY (returns true if there exist tuples (returned from nested WHERE clause) obey the condition!)
ALL (returns true if all tuples (returned from nested WHERE clause) obey the condition!)

```
SELECT *
FROM Sailors S
WHERE S.rating > ALL (SELECT S2.rating
                      FROM Sailors S2
                      WHERE S2.sname='Horatio')
```

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Division in SQL

Lancaster University

Find sailors who've reserved all boats.

EXISTS (returns true if the set is nonempty)

NOT EXISTS (returns true if the set is empty)

```
SELECT  S.sname
FROM  Sailors S
WHERE  NOT EXISTS
          ((SELECT  B.bid                    A
            FROM  Boats B)
           EXCEPT                            -
            (SELECT  R.bid                   B
             FROM  Reserves R
             WHERE  R.sid =  S.sid))
```

sid	*sname*	*rating*	*age*
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Figure 4.15 An Instance $S3$ of Sailors

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Figure 4.16 An Instance $R2$ of Reserves

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Figure 4.17 An Instance $B1$ of Boats

Set A= The set of all boats.
Set B= The set of all boats reserved by the current sailor S.sid
A-B is the boats that S.sid did not reserve. So if this set is empty, then return this s.sname!