Database Systems (ISYS1001/ISYS5008)

Lecture 5

Working with more than two tables

Updated: 25th August,2021

Discipline of Computing
School of Electrical Engineering, Computing and Mathematical Sciences (EECMS)

CRICOS Provide Code: 00301J

Copyright Warning

COMMONWEALTH OF AUSTRALIA
Copyright Regulation 1969

WARNING

This material has been copied and communicated to you by or on behalf of **Curtin University of Technology** pursuant to Part VB of the *Copyright Act 1968* (**the Act**)

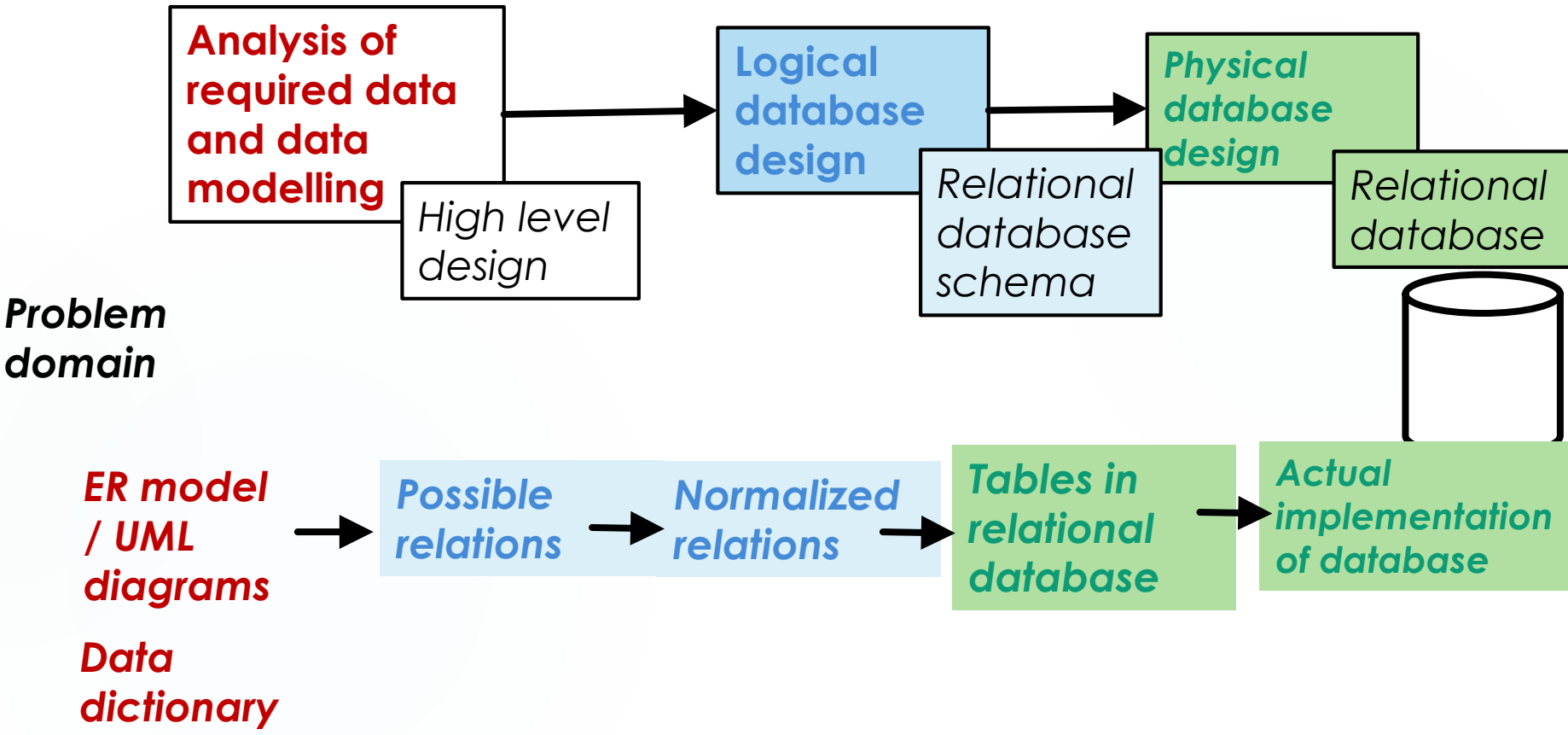
The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice

Learning outcomes

- Use different forms of joins to obtain information from more than one table.
- Explain the difference between different forms of joins
- Implement queries in MySQL with different forms of joins to retrieve data.
- Using MYSQL implement a relational schema with multiple tables and foreign key constraints.

Real world to database



We have looked at logical database design stage in the previous lecture.

Implementing the Schema using MYSQL

- You may have already implement some schema derived by you in the practical class.
- Once the relations are identified and the primary , foreign keys are known, converting the schema to SQL DDL statements is a straightforward task.
- Depend on the DBMS we are using, slightly different variations of the DDL command may be used.

- Example:

```
Student(studentID,firstName,lastName,courseName)
    FK courseName REF Course(courseName)
Course(courseName,CourseType)
```

Example:

- MySQL DDL statements for implementing the tables:

```
CREATE TABLE Courses(
    courseName VARCHAR(30) PRIMARY KEY,
    courseType VARCHAR(10)
);
CREATE TABLE Students(
    studentID VARCHAR(8) PRIMARY KEY,
    firstName VARCHAR(25) NOT NULL,
    lastName VARCHAR(25),
    courseName VARCHAR(30)
    CONSTRAINT fk_course
        FOREIGN KEY (courseName) REFERENCES Courses(courseName)
);
```

Student(studentID,firstName,lastName,courseName) FK courseName REF Course(courseName)

Course(courseName,CourseType)

Implementing the Schema using MYSQL

- Many other clauses to improve the performance, searching and avoiding anomalies are also included in CREATE TABLE statements.
- We will add these clauses to improve our relational database in next few lectures.

Queries using multiple relations

- So far, we have queried only one table at a time.
- In practical situations, we need to retrieve data from multiple tables at once.
- We may need to match up tables in a way that makes sense, and/or put constraints on what data we're choosing.
- In an ER diagram, entities and relations are linked by lines.
- In SQL tables, each relation has the keys of the entities that it links.
- We can use joins to query along these links.

Joining tables

- The command to join data from two tables together is called a JOIN.
- Think of the tables being written on paper or in a file – a JOIN operation creates a new table made out of both of the individual tables.
- The way that this is done depends on the sort of JOIN.
- Joins are derived based on relational algebra.

SQL Join variations

- COMMA JOIN
- CROSS JOIN
- THETA JOIN
- INNER JOIN
- OUTER JOIN
  - FULL OUTER JOIN
  - LEFT OUTER JOIN
  - RIGHT OUTER JOIN
- NATURAL JOIN
- In MySQL,
  - not all of these are supported
  - Joins are not implemented exactly the same way as mentioned in the SQL standard.

Sample relations

Students(studentID,firstName,lastName,country,courseID)

Courses(courseID,duration,dept)

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	C20
1002	Peter	Hess	Australia	NULL
1003	Elena	May	Malaysia	E10

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

Comma JOIN

- SQL provides a simple way to combine several relations in a single query: list each relation in the FROM clause. Then SELECT and WHERE clauses can refer to the attributes of the relations in the FROM clause.
- Example:

```
SELECT *
FROM Students, Courses;
```

This is simple, however product by itself is rarely useful

- In the above example , we are joining the tables but not putting any conditions on them. Every tuple from the first table is matched with every tuple from the second table. (Cartesian Product)
- Conditions can be included to reduce the tuples to look at.
- All tuples in the new table that don't match the condition will be discarded.
- Example:

```
SELECT *

FROM Students, Courses

WHERE country = 'Australia';
```

- This is called comma JOIN, because we simply list all of the tables involved and separate them with commas.
- Comma join is looking at the product of the relations.

Example

```
SELECT *
FROM Students, Courses;
```

3x3 =9 rows would be resulted.
All eight columns would be in the results.
This is simple, however product by itself is rarely useful

studentID	firstName	lastName	country	courseID	courseID	duration	dept
1001	Peter	Thompson	Australia	C20	C20	4	Computing
1002	Peter	Hess	Australia	NULL	C20	4	Computing
1003	Elena	May	Malaysia	E10	C20	4	Computing
1001	Peter	Thompson	Australia	C20	C21	3	Computing
1002	Peter	Hess	Australia	NULL	C21	3	Computing
1003	Elena	May	Malaysia	E10	C21	3	Computing
1001	Peter	Thompson	Australia	C20	E10	4	Engineering
1002	Peter	Hess	Australia	NULL	E10	4	Engineering
1003	Elena	May	Malaysia	E10	E10	4	Engineering

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	C20
1002	Peter	Hess	Australia	NULL
1003	Elena	May	Malaysia	E10

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

Example


```
SELECT *
FROM Students, Courses
WHERE country = 'Australia';
```

studentID	firstName	lastName	country	courseID	courseID	duration	dept
1001	Peter	Thompson	Australia	C20	C20	4	Computing
1002	Peter	Hess	Australia	NULL	C20	4	Computing
1001	Peter	Thompson	Australia	C20	C21	3	Computing
1002	Peter	Hess	Australia	NULL	C21	3	Computing
1001	Peter	Thompson	Australia	C20	E10	4	Engineering
1002	Peter	Hess	Australia	NULL	E10	4	Engineering

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	C20
1002	Peter	Hess	Australia	NULL
1003	Elena	May	Malaysia	E10

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

Examples

- We can retrieve only required attributes from both relations .
- Example:

```
SELECT studentID, firstName, dept
FROM Students, Courses
WHERE country= 'Australia' AND dept= 'Computing';
```

- Assume our `Students` relation is changed as follows:

```
Students(studentID,name,country,courseID)
Courses(courseID,name,duration,dept)
```

- Same attributes, `name` and `courseID` are in both relations
- If we have two or more relations with the same attribute names, SQL allows to place the *relation name , dot and the attribute name* to identifying the attribute without any ambiguity.

Example : Find student IDs, student names and the name of the course they have registered for all the students from Australia who have registered for a course.

```
SELECT studentID, Courses.name, Students.name
FROM Students, Courses
WHERE country= 'Australia' AND Students.courseID=Courses.courseID;
```

- We can write a query like follows as well, even when there is not ambiguity issues.

```
SELECT Students.studentID, Courses.name, Students.name
FROM Students, Courses
WHERE country='Australia' AND Students.courseID=Courses.courseID;
```

```
SELECT Students.studentID, Students.firstName, Students.name
FROM Students, Courses
WHERE Students.country='Australia' AND Students.courseID=Courses.courseID;;
```

Example: retrieve only required attributes from both relations

Students

studentID	firstName	lastName	country	courseID
1001	**Peter**	**Thompson**	**Australia**	**C20**
1002	**Peter**	**Hess**	**Australia**	**NULL**
1003	Elena	May	Malaysia	E10

Courses

courseID	duration	dept
C20	**4**	**Computing**
C21	**3**	**Computing**
E10	4	Engineering

```
SELECT StudentID, firstName, dept
FROM Students, Courses
WHERE country= 'Australia' AND dept= 'Computing';
```

studentID	firstName	dept
1001	Peter	Computing
1002	Peter	Computing
1001	Peter	Computing
1002	Peter	Computing

Example: retrieve data correctly when attributes are having same name

Students

studentID	name	country	courseID
1001	**Peter Thompson**	**Australia**	**C20**
1002	Peter Hess	Australia	NULL
1003	Elena May	Malaysia	E10

Courses

courseID	name	duration	dept
C20	**BSc(Comp.)**	**4**	**Computing**
C21	BSc(IT)	3	Computing
E10	BEng	4	Engineering

```
SELECT studentID, Courses.name, Students.name
FROM Students, Courses
WHERE country= 'Australia' AND Students.courseID=Courses.courseID;
```

studentID	name	name
1001	Peter Thompson	BSc(Comp.)

Example : Find student IDs, student names and the name of the course they have registered for all the students from Australia who have registered for a course.

Use of alias as relation names

- We may use a short name for the name of the relations in our queries.
- Example:

```
SELECT S1.studentID, C1.name, S1.name
FROM Students S1, Courses C1
WHERE country = 'Australia' AND S1.courseID=C1.courseID;
```

- Example: We may use AS keyword also but it is not mandatory.

```
SELECT S1.studentID, C1.name, S1.name
FROM Students AS S1, Courses AS C1
WHERE country = 'Australia' AND S1.courseID=C1.courseID;
```

Use of Comma Join style

- In general, you can use the comma JOIN properly to query several relations.
- MySQL will simply optimize it for whatever you're doing.
- However, there is some debate about the use of comma JOINs in the industry. A strong feeling is that you should explicitly state your JOIN type.
- Though simple and quick to write has precedence and style issues
- **It is encouraged not to use comma joins as it does not specify the type of join. Therefore, better to avoid using comma joins and use other forms to get the same results.**
- For this unit**, do not use a comma JOIN in your queries**.
- *There are many other ways we can obtain the same results of comma joins , with more meaningful join information,*

CROSS Join

- This is the simplest form of join. We can get the same results as comma join without conditions.
- Results of cross join of two relations would be all the columns of the both the relations.

Example : Consider the following two relations.

```
Student(studentID,firstName,lastName,country,courseID)

Courses(courseID,duration,dept)
```

- Then, cross join can be written by putting CROSS JOIN between the two relation names:

```
Students CROSS JOIN Courses
```

- Results of the above join would be eight column relation with all the attributes of `Students` and all the attributes of `Courses.`
- Same as Comma join without conditions.

Example : CROSS JOIN

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	C20
1002	Peter	Hess	Australia	NULL
1003	Elena	May	Malaysia	E10

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

```
Select *
FROM Student CROSS JOIN Courses;
```

3x3 =9 rows would be resulted
This is simple, however product by itself is rarely useful

studentID	firstName	lastName	country	courseID	courseID	duration	dept
1001	Peter	Thompson	Australia	C20	C20	4	Computing
1002	Peter	Hess	Australia	NULL	C20	4	Computing
1003	Elena	May	Malaysia	E10	C20	4	Computing
1001	Peter	Thompson	Australia	C20	C21	3	Computing
1002	Peter	Hess	Australia	NULL	C21	3	Computing
1003	Elena	May	Malaysia	E10	C21	3	Computing
1001	Peter	Thompson	Australia	C20	E10	4	Engineering
1002	Peter	Hess	Australia	NULL	E10	4	Engineering
1003	Elena	May	Malaysia	E10	E10	4	Engineering

Theta JOIN

- Theta join use ON keyword.
- We put JOIN between two relation names and follow them by ON and a condition.

```
<Relation1> JOIN <Relation2> ON <Condition>
```

- Only the product of the two relations R1 and R2 adhering to the condition given in ON clause will be resulted.
- **Same as replacing the WHERE condition in comma join with an ON condition, which is slightly more specific.**
- *Example : Find student IDs, student's first name and the courseID for all the students registered for a course and from Malaysia.*

```
SELECT studentID, firstname, Students.courseID
FROM Students JOIN Courses ON
    country = 'Malaysia' AND Students.courseID=Courses.courseID;
```

Example :Theta JOIN

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	C20
1002	Peter	Hess	Australia	NULL
1003	**Elena**	**May**	**Malaysia**	**E10**

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

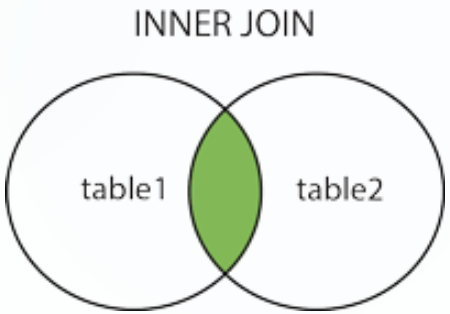
MySQL:

```
SELECT studentID, firstname, Students.courseID
FROM Students JOIN Courses ON
    country = 'Malaysia' AND Students.courseID=Courses.courseID;
```

studentID	firstName	courseID
1003	Elena	E10

INNER Join

- Equi JOIN or INNER Join
- X [INNER] JOIN Y ON C
  - Similar to the previous joins, where the condition C must be one that compares based on equality.
  - Not actually a different type of join, but is referred to specifically because it is used so often.
- Term INNER JOIN make it clear and distinguish the join from other forms



Syntactically same as JOIN
INNER JOIN is clear than JOIN

Image source: https://www.w3schools.com/sql/sql_join_right.asp

Example : INNER JOIN

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	C20
1002	Peter	Hess	Australia	NULL
1003	**Elena**	**May**	**Malaysia**	**E10**

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

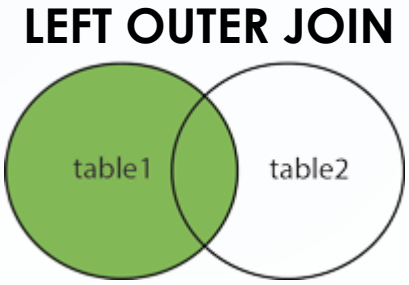
MySQL:

```
SELECT studentID, firstname, Students.courseID
FROM Students INNER JOIN Courses ON
    country = 'Malaysia' AND Students.courseID=Courses.courseID;
```

studentID	firstName	courseID
1003	Elena	E10

OUTER JOINS

- Suppose we join *R* and *S*.
- A tuple of *R* that has no tuple of *S* with which it joins is said to be *dangling*.
  - Similarly for a tuple of *S*.
- An outer join preserves dangling tuples in both tables by padding them NULL.
- A left or right join preserves only the dangling tuples of the left and right tables, respectively.
- Note that the full outer join is not available in MySQL.



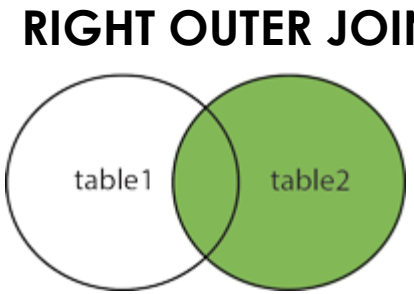


Image source: https://www.w3schools.com/sql/sql_join_right.asp

[LEFT/RIGHT] OUTER JOINS

- LEFT OUTER JOIN
  - Resulting all the tuples of the first relation listed (left table) and tuples from the second relation (right table)with matching condition, and any dangling tuples with NULL values.
- RIGHT OUTER JOIN
  - Resulting all the tuples of the second relation listed (righty table) and tuples from the first relation (left table)with matching condition, and any dangling tuples with NULL values.

Example : LEFT OUTER JOIN

Students

studentID	firstName	lastName	country	courseID
1001	**Peter**	**Thompson**	**Australia**	**C20**
1002	**Peter**	**Hess**	**Australia**	**NULL**
1003	**Elena**	**May**	**Malaysia**	**E10**

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

MySQL:

```
SELECT *
FROM Students LEFT OUTER JOIN Courses ON
    country = 'Malaysia' AND Students.courseID=Courses.courseID;
```

studentID	firstName	lastName	country	courseID	courseID	duration	dept
1003	Elena	May	Malaysia	E10	E10	4	Engineering
1001	Peter	Thompson	Australia	C20	**NULL**	**NULL**	**NULL**
1002	Peter	Hess	Australia	NULL	**NULL**	**NULL**	**NULL**

Example : RIGHT OUTER JOIN

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	C20
1002	Peter	Hess	Australia	NULL
1003	Elena	May	Malaysia	E10

Courses

courseID	duration	dept
C20	**4**	**Computing**
C21	**3**	**Computing**
E10	**4**	**Engineering**

MySQL:

```
SELECT *
FROM Students RIGHT OUTER JOIN Courses ON
    country = 'Malaysia' AND Students.courseID=Courses.courseID;
```

studentID	firstName	lastName	country	courseID	courseID	duration	dept
1003	Elena	May	Malaysia	E10	E10	4	Engineering
NULL	**NULL**	**NULL**	**NULL**	**NULL**	C20	4	Computing
NULL	**NULL**	**NULL**	**NULL**	**NULL**	C21	3	Computing

NATURAL Join

- If we have same names as attribute names, without referring to them in a separate clause, we can join the two relations through the common names using NATURAL JOIN.

```
<relation1> NATURAL JOIN <relation2>
```

- Differs from Theta join as:
  - Join condition is that all pairs of the attributes from the two relations having a common name are equated and there are no other conditions.
  - Removing one copy of each pair of equated attributes.
- Called *natural* JOIN because it is the natural way to join two tables; Uses the default constraint that attributes with the same name must be equal
- It is often worthwhile to name an attribute in such a way that you can use NATURAL JOINs.
- More efficient than using a theta/inner equi join
- Considered bad coding by some because the condition isn't made explicit

Example : NATURAL JOIN

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	**C20**
1002	Peter	Hess	Australia	NULL
1003	Elena	May	Malaysia	**E10**

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

MySQL:

```
SELECT *
FROM Students NATURAL JOIN Courses;
```

Results show attribute CourseID and all other attributes appear in one or the other relation

courseID	studentID	firstName	lastName	country	duration	dept
C20	1001	Peter	Thompson	Australia	4	Computing
E10	1003	Elena	May	Malaysia	4	Computing

```
SELECT *
FROM Courses NATURAL JOIN Students;
```

What would be the results of the above query ?

NATURAL [LEFT/RIGHT] JOIN

- MySQL combines NATURAL JOIN with LEFT or RIGHT keywords to obtain all tuples from one relation and only the tuples from the other relation which have same attribute values as that relation.
- NATURAL LEFT JOIN
  - All the tuples of the first relation listed (left table) and only the tuples from the second relation (right table) having same values as the first relation(left table) for the common attribute.
- NATURAL RIGHT JOIN
  - All the tuples of the second relation listed (right table) and only the tuples from the first relation (left table) having same values as the second relation (right table) for the common attribute.

Example : NATURAL JOIN

Students

studentID	firstName	lastName	country	courseID
1001	Peter	Thompson	Australia	C20
1002	Peter	Hess	Australia	NULL
1003	Elena	May	Malaysia	E10

Courses

courseID	duration	dept
C20	4	Computing
C21	3	Computing
E10	4	Engineering

MySQL:

```
SELECT *
FROM Students NATURAL LEFT JOIN Courses;
```

courseID	studentID	firstName	lastName	country	duration	dept
C20	1001	Peter	Thompson	Australia	4	Computing
E10	1003	Elena	May	Malaysia	4	Computing
NULL	1002	Peter	Hess	Australia	NULL	NULL

```
SELECT *
FROM Students NATURAL RIGHT JOIN Courses;
```

courseID	duration	dept	studentID	firstName	lastName	country
C20	4	Computing	1001	Peter	Thompson	Australia
E10	4	Engineering	1003	Elena	May	Malaysia
C21	3	Computing	NULL	NULL	NULL	NULL

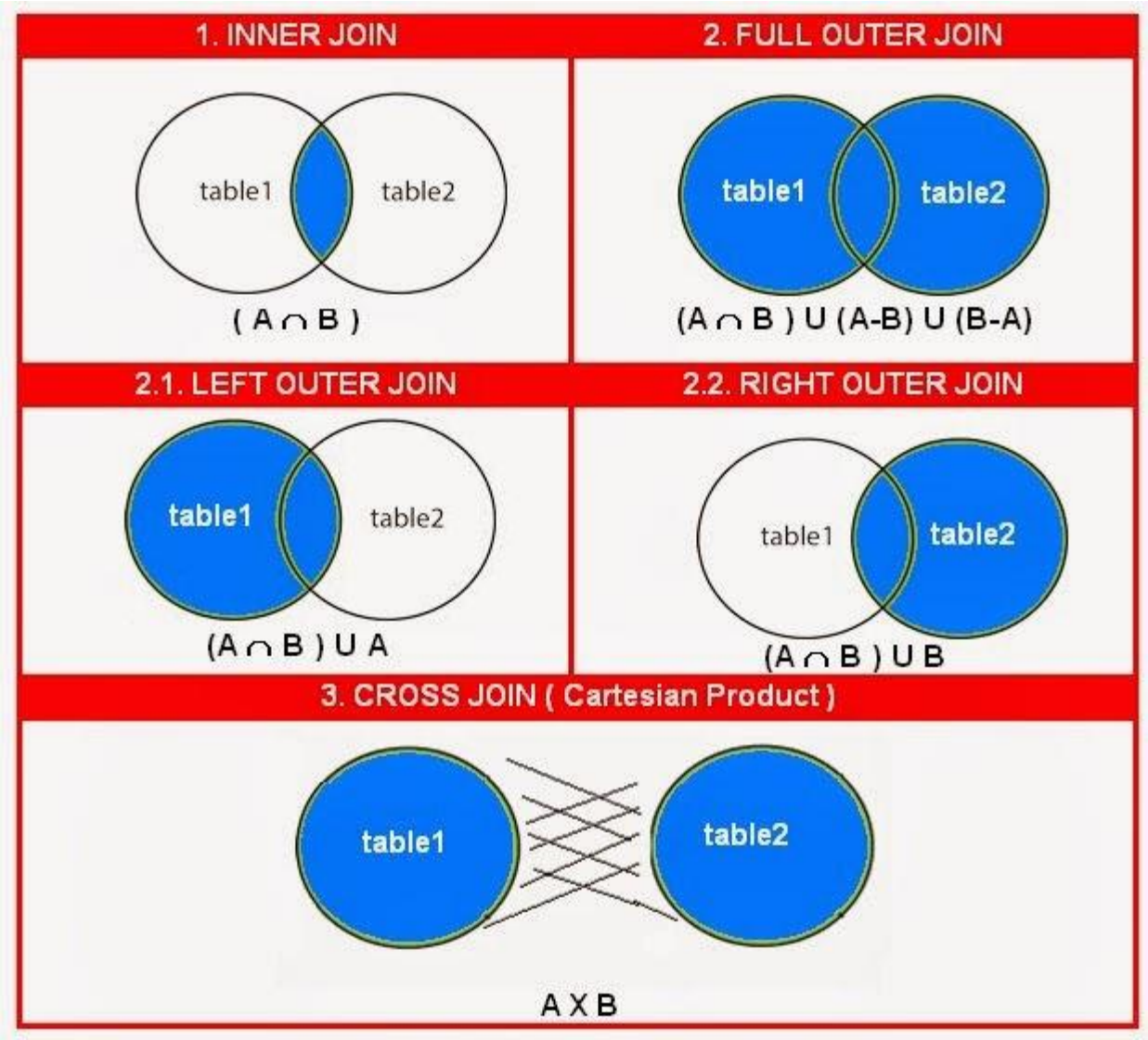
MySQL Joins - revisit

- Comma JOIN: X, Y
  - Basic join that is quick to write but has precedence and style issues.
  - DO NOT use it in this unit.
- Cross JOIN: X CROSS JOIN Y ON C
  - Same as comma join
  - Not commonly used
- Theta JOIN : X JOIN Y ON C
  - Just like the comma JOIN with constraint C.
  - Constraint C can be a comparison using >,<, >=, <= <>
- Inner JOIN : X INNER JOIN Y ON C
  - Again, just like the comma (and theta) joins.
  - Good coding because it specifies the type of join.
  - Equi Inner Join : when the condition is = , it is called a equi join
- NATURAL JOIN : X NATURAL JOIN Y
  - common name are equated and there are no other conditions
  - A specific implementation of the equi inner join
  - Can have [lEFT/Right] variations also

MySQL Joins

- OUTER JOIN : X [LEFT /RIGHT] OUTER JOIN ON C
  - Full Outer JOIN - X FULL OUTER JOIN Y ON C
    - Equivalent to a THETA JOIN except that an OUTER JOIN keeps dangling tuples
    - Not implemented in MySQL
  - Left or Right JOIN – X LEFT JOIN Y ON C
    - These joins are like the Outer JOIN, but only preserve the dangling tuples from the table on the left or right.
    - Do work in MySQL.
    - Can be used (with UNION) to make a FULL OUTER JOIN

Graphical Representation of Joins

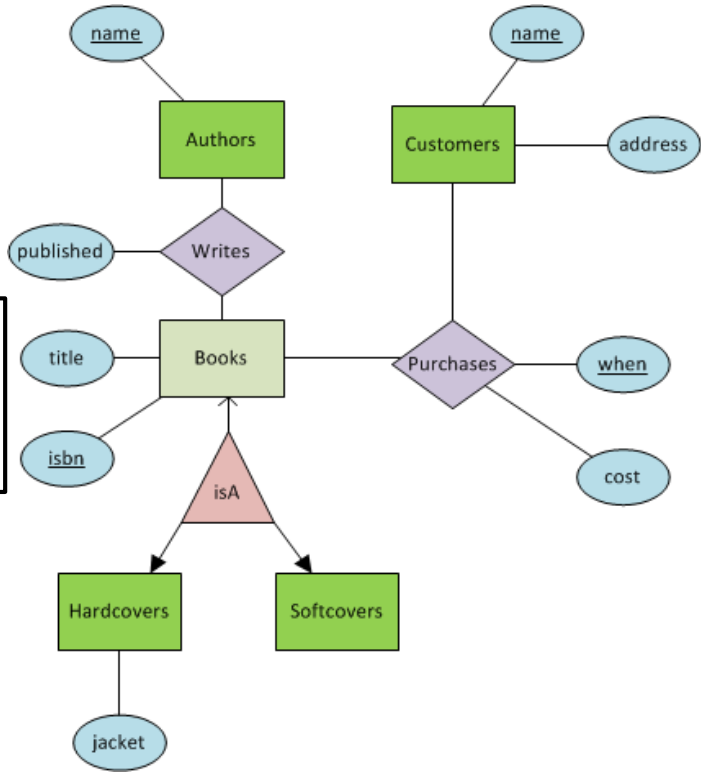


Taken from https://stackoverflow.com/questions/17946221/sql-join-and-different-types-of-joins

Exercise:

- Find the name and the address of the customers who have purchased a particular book (9780142001196).
- In order to get information from both Customers and Purchases tables, we can use a JOIN.

```
SELECT Customer.name, address
    FROM Purchases NATURAL JOIN Customers
    WHERE book=9780142001196;
```



- In this case, the key name is in both tables (and has the same name) so we can use a NATURAL JOIN.
- We could also use:

Purchases(name, bookid,when, cost)

```
SELECT Customer.name, address
    FROM Purchases INNER JOIN Customers
    ON Purchases.name = Customers.name AND book=9780142001196;
```

Queries comparing two tuples in the same relation

- When there are two tuples in the same relation to compare, table is joined with itself.
- We can list the relation as many times we need in the FROM clause. However we need a way to identify the each occurrence of the relation .
- In such a case, we can use another name for the relation (“alias” or tuple variable)
- Example: Consider the `Students` relation and find student IDs, country of the students from the same country :

```
SELECT S1.studentID S1ID, S2.studentID S2ID, S1.country
FROM Students S1, Students S2
WHERE S1.country=S2.country AND S1.studentID<>S2.studentID;
```

- Same query can be written with JOIN ON as follows:

```
SELECT S1.studentID S1ID, S2.studentID S2ID, S1.country
FROM Students S1 JOIN Students S2 ON S1.country=S2.country
WHERE S1.studentID<>S2.studentID;
```

S1ID	S2ID	country
1002	1001	Australia
1001	1002	Australia

MySQL Query Modification

- One of the main purposes of a DBMS is to display data in a sensible way.
- This means there are several modifications that you can make to the way that queries display results.
- One of the simplest is to sort the output of a query.
- Summarizing results also is useful.

Emp table schema used in next few examples

Name	Null?	Type
EMPNO	NOT NULL	CHAR(6)
FIRSTNAME	NOT NULL	VARCHAR2(12)
MIDINIT	NOT NULL	CHAR(1)
LASTNAME	NOT NULL	VARCHAR2(15)
WORKDEPT		CHAR(3)
PHONENO		CHAR(4)
HIREDATE		DATE
JOB		CHAR(8)
EDLEVEL		DECIMAL(2)
SEX		CHAR(1)
BIRTHDATE		DATE
SALARY		DECIMAL(8,2)
BONUS		DECIMAL(8,2)
COMM		DECIMAL(8,2)

Practical-3 worksheet

ORDER BY

- Practical3, Task3.Q1: *Display the last name, work department and salary of all employees who get a salary of $100,000 or more per year.*
- *T*he answer creates a messy looking table.
- When generating long tables, it's often better to sort the output.
- Sorting can be done using ORDER BY clause

```
SELECT lastname, workdept, salary
   FROM Emp
   WHERE salary >= 100000
   ORDER BY salary;
```

- The results will now be displayed in order of increasing salary.

ORDER BY

- General format:

```
SELECT <attributes>
   [FROM <tables>
   [WHERE <condition>
   [ORDER BY <attributes> [DESC]
]]];
```

- We can order by multiple attributes, for example:

```
ORDER BY lastname, firstname
```

- By default, the attribute named is sorted in ascending order (from small to large). Include the DESC keyword to reverse this.

GROUP BY

- Sometimes we want to merge query results on certain attributes, and perform functions on others (summary results)
- SQL handles this through grouping.
- For example, *what if we wanted to show the minimum salary for each type of job in the Emp database?*

```
SELECT job, MIN(salary)
FROM Emp
GROUP BY job;
```

- The result is one clause per job, with the minimum of all the salaries associated with that job beside it.
- Min is an *aggregate function*.

GROUP BY

- In SQL, if there is an aggregation in a query, then all attributes must either be aggregate attributes (*e.g.*, MIN(salary)) or be grouped in the GROUP BY line.
- This query is illegal in SQL:

```
SELECT job, MIN(salary), bonus
   FROM Emp
   GROUP BY job;
```

- However, MySQL allows it – after all, it makes sense in some cases.

Aggregation Functions

- There are several functions that you can use for aggregation:
  - `MAX`
  - `MIN`
  - `SUM`
  - `COUNT`
  - `AVG` (for the average)
- It is always need to specify which attribute you are applying the function to:
  - *e.g.*, `MIN(salary), MAX(age)`
- We may use `AS` to rename the column when aggregating.

Auto-Sort

- In SQL, any attribute that is grouped is automatically sorted.

```
SELECT job, MIN(salary)
 FROM Emp
 GROUP BY job;
```

- The output table has the *job* column sorted in ascending order.
- In MySQL, sorting can be turned off :

```
SELECT job, MIN(salary)
    FROM Emp
    GROUP BY job
    ORDER BY NULL;
```

- We can also choose to sort in descending order:

```
SELECT job, MIN(salary)
    FROM Emp
    GROUP BY job DESC
```

Summary

- There are many ways we can retrieve data from more than one table; SQL offer several Join types.
- Comma join is simple to use but not a good choice.
- INNER JOIN, OUTER [LEFT/RIGHT] JOIN, NATURAL JOIN are common joins useful in MYSQL
- Theta –join and CROSS JOIN are also avialble.
- GROUP BY can be used to create summary outputs of results.
- Aggregate functions are used for grouping (MIN(),,MAX(), SUM(),COUNT(),AVG())
- Sorting of the results can be done using ORDER BY clause.
- Sort in descending order is possible by combining DESC with ORDER BY clause

Happy Database systems

Next week : Table manipulation

Practical Test -2 will be held in the next week