Database Systems 2

Lecture 6

Advanced SQL 2

Data Manipulation

Lecture - Objectives

- SELECT statements
- Joins
- Aggregate Functions

Relational Algebra

In computer science, relational algebra is an offshoot of first-order logic and of algebra of sets concerned with operations over finitary relations, usually made more convenient to work with by identifying the components of a tuple by a name (called attribute) rather than by a numeric column index, which is called a relation in database terminology.

The main application of relational algebra is providing a theoretical foundation for relational databases, particularly query languages for such databases, chief among which is SQL.

From Wikipedia, the free encyclopedia

Table

Row / Record

Field

In short, one or more tables are fed into an operation, and one table comes out.

Relational Operators

Projection

- vertical subset of columns to create a new table

Restriction

- horizontal subset of tuples to create a new table

Join

- join two or more relations together to create a third table

Union

- adding two tables of the same structure to create a third table

Intersect

- extracting common records from two tables to create a third table

Student

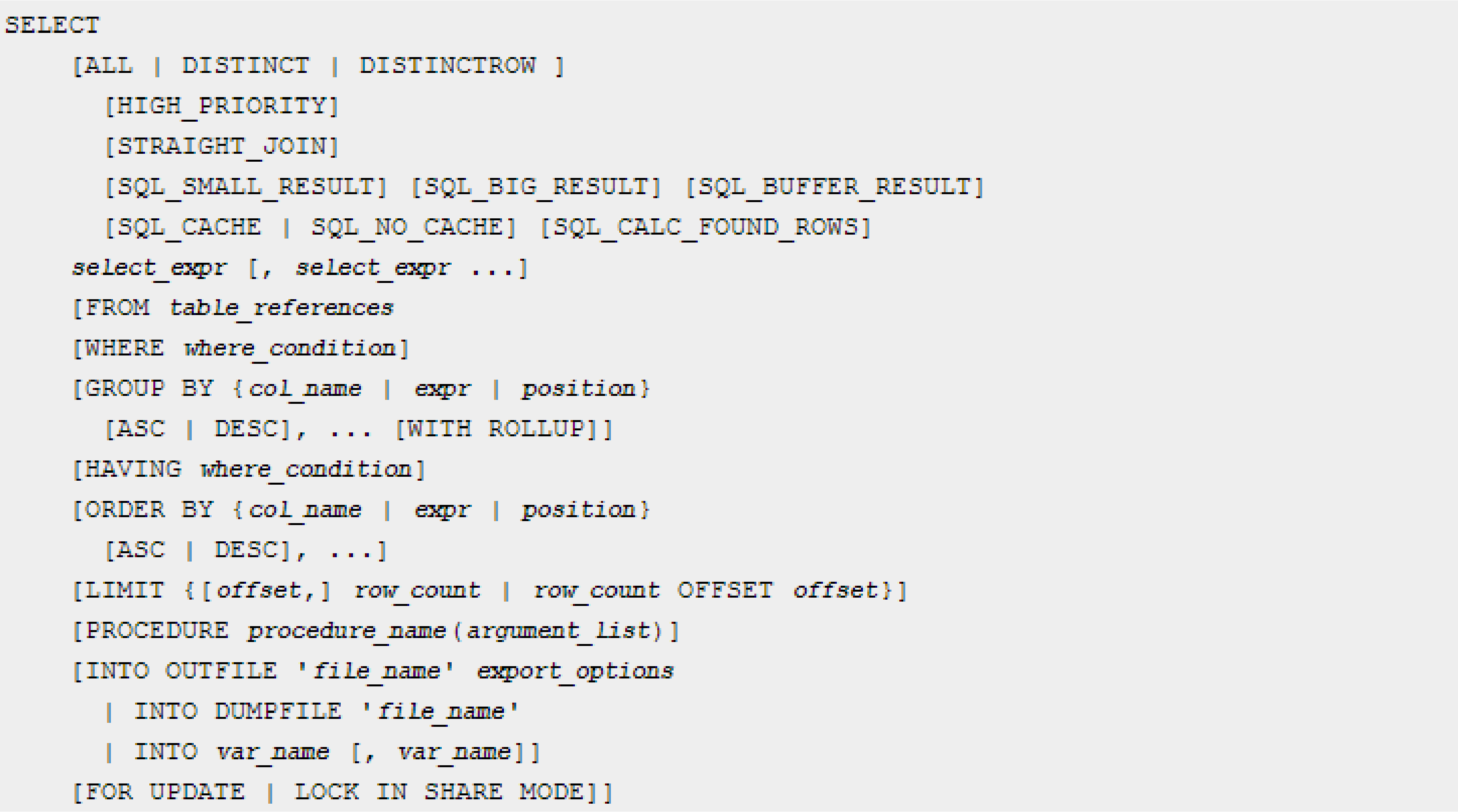
ID	Name	Add1	Add2	Pcode	Course_ID*	Year	DOB	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
4	Johnson	11 Ashley Road	Poole	BH12	MCS	1	09-Mar-77	69	F
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F
12	Swift	6 Church St	Poole	BH9	COMP	1	30-Apr-72	12	M
13	Chambers	98 High St	Poole	BH5	COMP	2	06-Aug-75	78	F

Course

Course_ID	Name	Years	Leader	Type
ICS	Internet Computing	4	Paul	Degree
COMP	Computing	4	Mike	Degree
MCS	Multimedia	4	Paul	Degree
BIT	Business	4	David	Degree

The SELECT statement

SELECT Syntax



```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr ...]
    [FROM table_references
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [INTO OUTFILE 'file_name' export_options
      | INTO DUMPFILE 'file_name'
      | INTO var_name [, var_name]]
    [FOR UPDATE | LOCK IN SHARE MODE]]
```

SELECT Queries

The SELECT keyword is used to create queries that retrieve data from a database.

To retrieve all the data from a relation (all columns, all rows):

```
SELECT *
FROM tablename ;
```

SELECT clause lists the attributes

FROM clause lists the tables to be used in the query

Example Output

```
SELECT *
FROM student;
```

ID	Name	Add1	Add2	Pcode	Course_ID*	Year	DOB	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
4	Johnson	11 Ashley Road	Poole	BH12	MCS	1	09-Mar-77	69	F
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F
12	Swift	6 Church St	Poole	BH9	COMP	1	30-Apr-72	12	M
13	Chambers	98 High St	Poole	BH5	COMP	2	06-Aug-75	78	F

Projection Query

Specify which columns to retrieve from which table:

```
SELECT name, add1
FROM student;
```

```
SELECT Course_ID, leader
FROM course;
```

Name	Add1
Jones	10 Old Street
Bloggs	12 Alder Way
Johnson	11 Ashley Road
Walker	99 Oldcott Road
Harrison	10 Daly Road
Swift	6 Church Street
Chambers	98 High Street

Course_ID	Leader
ICS	Paul
COMP	Mike
MCS	Paul
BIT	David

Ordering the results of a query

In ASCENDING order :

```
SELECT ID, name
FROM student
ORDER BY name asc;
```

ID	Name
3	Bloggs
13	Chambers
11	Harrison
4	Johnson
1	Jones
12	Swift
5	Walker

In DESCENDING order:

```
SELECT ID, name
FROM student
ORDER BY name desc;
```

ID	Name
5	Walker
12	Swift
1	Jones
4	Johnson
11	Harrison
13	Chambers
3	Bloggs

Restriction Query

To select only rows satisfying a particular condition:

```
SELECT *
FROM student
WHERE add2 = 'Bournemouth';
```

ID	Name	Add1	Add2	Pcode	Course_ID	Year	Dob	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F

Restriction Query

To link WHERE conditions using AND

```
SELECT *
FROM student
WHERE add2 = 'Bournemouth'
AND course_id= 'BIT';
```

ID	Name	Add1	Add2	Pcode	Course _ID	Year	Dob	Mark	Gender
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F

Nested Query(Projection and restrict)

```
SELECT name, dob
FROM student
WHERE add3 = "Bournemouth";
```

Name	Dob
Jones	03-Jun-82
Bloggs	05-Sep-80
Walker	15-May-82
Harrison	24-Jul-70

```
SELECT name, dob
FROM student
WHERE add3 = 'Bournemouth'
AND course_id = 'BIT';
```

Name	Dob
Bloggs	05-Sep-80
Harrison	24-Jul-70

WHERE clause operators

Relational operators

=, >, >=, <, <=, !=, <>

eg *WHERE age < 18*

Boolean operators

AND, OR, NOT

eg *WHERE name = “Smith" OR name = “Jones"*

Other

BETWEEN, LIKE, IN, IS

eg *WHERE year BETWEEN 2 AND 4*

eg *WHERE add1 LIKE "%Alder%"*

Nested Queries (Project, Restrict and Join)

```
SELECT student.name, course.course_name
FROM student, course
WHERE student.course_id = course.course_id
AND add3 = 'Bournemouth';
```

Name	Course_Name
Jones	Internet Computing
Bloggs	Business
Walker	Internet Computing
Harrison	Business

Again, note use of table names to clarify which table the data is coming from

Union

Merge together records from two tables:

```
 SELECT ID, name, course_id
 FROM student
UNION
  SELECT ID, name, course_id
  FROM old_student
```

ID	Name	Course_ID
1	Jones	ICS
3	Bloggs	BIT
4	Johnson	MCS
5	Walker	ICS
11	Harrison	BIT
12	Swift	COMP
13	Chambers	COMP
2	Smith	BIT
7	Shangali	ICS
9	Harris	ICS
10	Swift	ICS
14	Robinson	MCS

Intersection

Records that are common to both tables:

```
 SELECT ID, name, course_id
 FROM student
INTERSECT
 SELECT ID, name, course_id
 FROM old_student
```

Doesn’t work in Access, but does in other relational databases eg Oracle.

JOINS

Joining Tables

It is possible to define queries that access more than one table. You are said to be making a join between two tables if you do this.

To join two tables together, there must be a common field – *a primary key in one which is a foreign key in the other.*

If there isn't, you may get a very unexpected result.

The Cartesian Product

You will get an output table consisting of every possible combination of rows from the two input tables.

That is exactly the what you get when you type:

```
SELECT  *
FROM emp, dept
```

Joins

In order to avoid the Cartesian product we use Joins

There are two main categories of join:

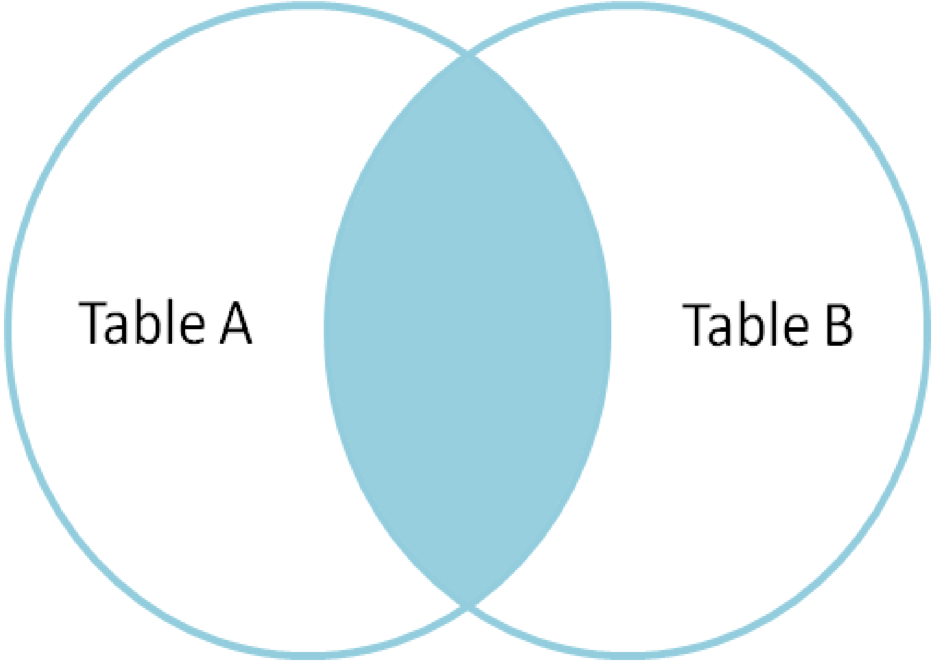
- **Inner Joins**
- **Outer Joins (Left, Right, Full)**

Inner Joins

An inner join is one in which a row is output only when there is at least one row in each of the input tables which matches the condition.

```
SELECT  *
FROM emp, dept
WHERE emp.deptno = dept.deptno;
```

Inner Joins



Inner Joins Note:

- The FROM statement specifies both tables.
- The WHERE clause is the thing that joins the tables together (order is not important).
- How the table name and the dot is used in front of the field name in the WHERE clause, to avoid ambiguity.
- Due to the *, all fields from both tables are displayed, including both deptno fields.

Table Aliases

```
SELECT empno, ename, dept.deptno, dname
   FROM emp, dept
   WHERE emp.deptno = dept.deptno
   ORDER BY dept.deptno;
```

could be rewritten:

```
SELECT empno, ename, d.deptno, dname
   FROM emp e, dept d
   WHERE e.deptno = d.deptno
   ORDER BY d.deptno;
```

A Join Between 3 Tables

```
SELECT employee.emp_name, room.room_no, telephone.extension
   FROM employee, room, telephone
   WHERE employee.room_no = room.room_no
   AND room.room_no = telephone.room_no;
```

You can create queries which join as many tables as you want.

If your queries start to get a bit unwieldy, it may be a sign that you need to rethink the relationships in your database.

Other Ways of Specifying a Join

Instead of using the WHERE clause, you can use the JOIN and ON clauses:

```
SELECT emp_no, emp_name, telephone.room_no, extension
   FROM employee JOIN telephone
   ON employee.room_no = telephone.room_no;
```

OR, if the foreign key field has the same name as the primary key field

```
SELECT emp_no, emp_name, room_no, extension
   FROM employee JOIN telephone
   USING (room_no);
```

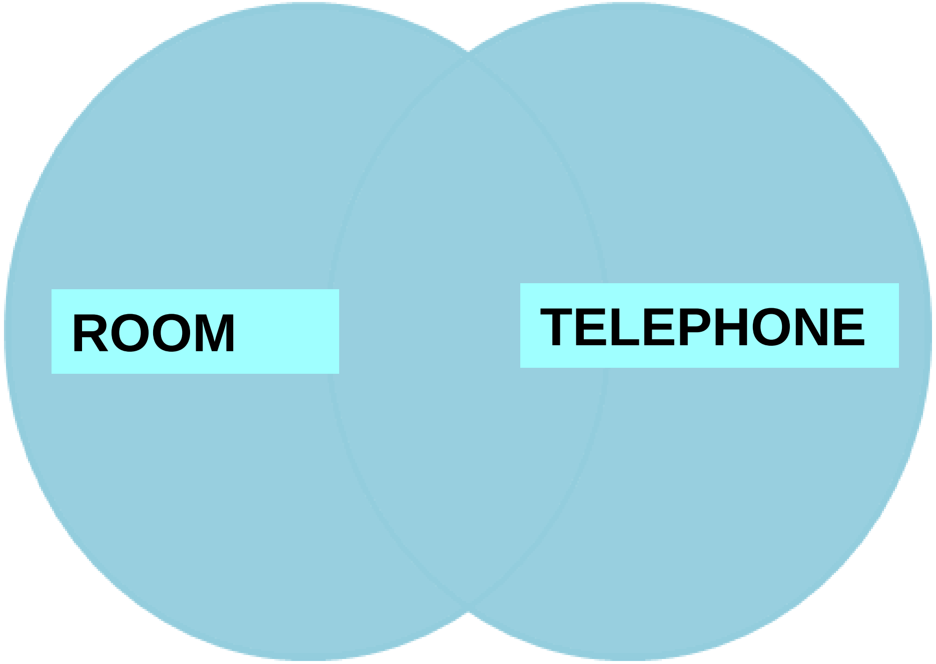
Outer Joins

An Outer join is one in which a row in one of input tables will produce a row in the output table even if there isn’t a matching row in the other input table.

In these types of join, the order in which the tables are listed <u>is</u> important.

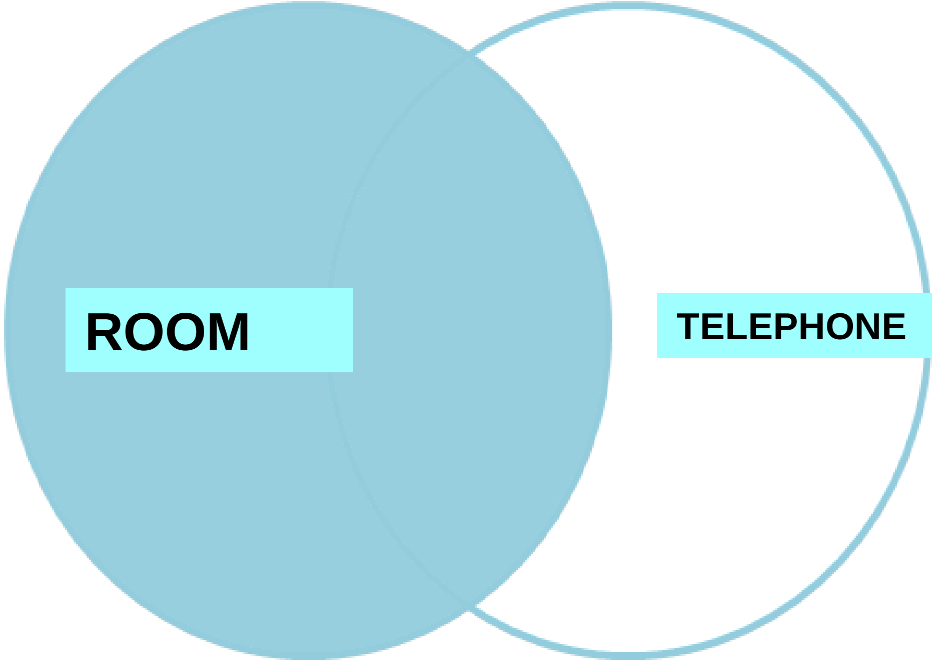
Full Outer Joins

```
SELECT room_no, capacity, extension
FROM room FULL OUTER JOIN telephone
USING (room_no);
```



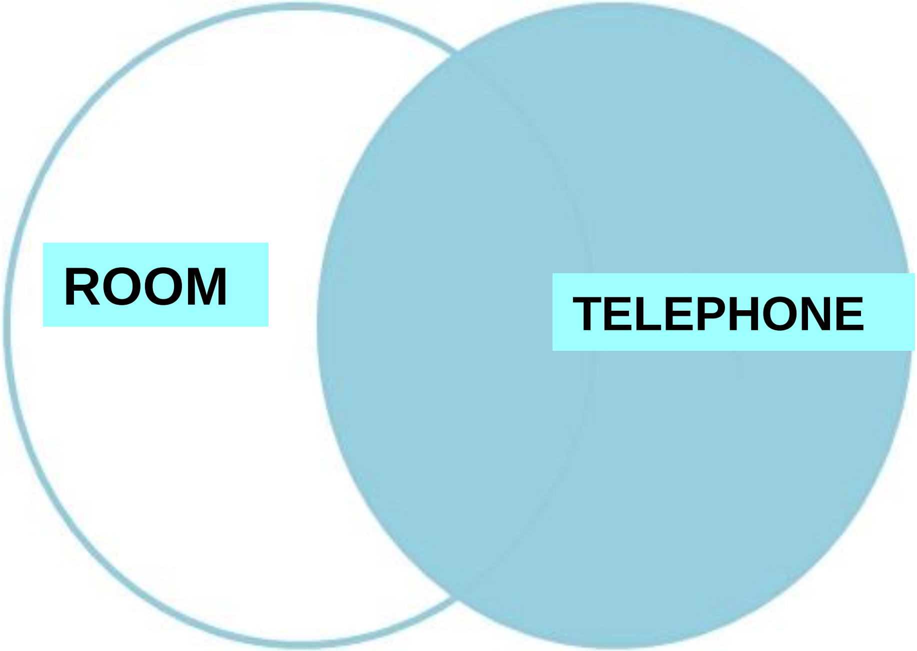
Left Outer Joins

```
SELECT room_no, capacity, extension
FROM room LEFT OUTER JOIN telephone
USING (room_no);
```



Right Outer Joins

```
SELECT room_no, capacity, extension
FROM room RIGHT OUTER JOIN telephone
USING (room_no);
```



Example Files

room

<u>room_no</u>	capacity
R1	5
R2	4
R3	1
R4	3

telephone

<u>extension</u>	location	room_no*
217	desk	R3
218	wall	R4
219	desk	R5
350	wall	R6

Inner

room_no	capacity	extension
R3	1	217
R4	3	218

Left Outer

room_no	capacity	extension
R1	5	NULL
R2	4	NULL
R3	1	217
R4	3	218

Right Outer

room_no	capacity	extension
R3	1	217
R4	3	218
R5	NULL	219
R6	NULL	350

Full Outer

room_no	capacity	extension
R1	5	NULL
R2	4	NULL
R3	1	217
R4	3	218
R5	NULL	219
R6	NULL	350

Calculations
Aggregate Functions

Calculations

Multiplication

```
SELECT ID, name, mark, mark*1.05 AS New_Mark
FROM student;
```

ID	Name	Mark	New_Mark
1	Jones	66	69.3
3	Bloggs	40	42
4	Johnson	69	72.45
5	Walker	78	81.9
11	Harrison	85	89.25
12	Swift	12	12.6
13	Chambers	78	81.9

Aggregate functions

These functions operate on **a number of rows (*)** to produce summary information, taking a column name as argument.

```
SELECT Count(*) AS Females
FROM student
WHERE student.gender="F";
```

Females
3

Other aggregate functions include: MAX, MIN, SUM

Aggregate Functions Example

Student

ID	Name	Add1	Add2	Pcode	Course_ID*	Year	DOB	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
4	Johnson	11 Ashley Road	Poole	BH12	MCS	1	09-Mar-77	69	F
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F
12	Swift	6 Church St	Poole	BH9	COMP	1	30-Apr-72	12	M
13	Chambers	98 High St	Poole	BH5	COMP	2	06-Aug-75	78	F

61.1

What would happen if I tried this?

```
SELECT AVG(mark) AS AvgOfmark
FROM student;
```

AvgOfmark
61.1

Add an extra field

Student

ID	Name	Add1	Add2	Pcode	Course_ID*	Year	DOB	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
4	Johnson	11 Ashley Road	Poole	BH12	MCS	1	09-Mar-77	69	F
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F
12	Swift	6 Church St	Poole	BH9	COMP	1	30-Apr-72	12	M
13	Chambers	98 High St	Poole	BH5	COMP	2	06-Aug-75	78	F

61.1

Add an extra field:

```
SELECT Course_ID, Avg(mark) AS AvgOfmark
FROM student;
```

Error message

Why?

Student

ID	Name	Add1	Add2	Pcode	Course_ID*	Year	DOB	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
4	Johnson	11 Ashley Road	Poole	BH12	MCS	1	09-Mar-77	69	F
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F
12	Swift	6 Church St	Poole	BH9	COMP	1	30-Apr-72	12	M
13	Chambers	98 High St	Poole	BH5	COMP	2	06-Aug-75	78	F

COMP MCS ICS BIT

61.1

```
SELECT Course_ID, Avg(mark) AS AvgOfmark
FROM student;
```

Error message

Use GROUP BY

Student

ID	Name	Add1	Add2	Pcode	Course_ID*	Year	DOB	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
4	Johnson	11 Ashley Road	Poole	BH12	MCS	1	09-Mar-77	69	F
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F
12	Swift	6 Church St	Poole	BH9	COMP	1	30-Apr-72	12	M
13	Chambers	98 High St	Poole	BH5	COMP	2	06-Aug-75	78	F

COMP	45
MCS	69
ICS	72
BIT	62.5

```
SELECT Course_ID, Avg(mark) AS AvgOfmark
FROM student
GROUP BY Course_ID;
```

Does this make sense?

Student

ID	Name	Add1	Add2	Pcode	Course_ID*	Year	DOB	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
4	Johnson	11 Ashley Road	Poole	BH12	MCS	1	09-Mar-77	69	F
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F
12	Swift	6 Church St	Poole	BH9	COMP	1	30-Apr-72	12	M
13	Chambers	98 High St	Poole	BH5	COMP	2	06-Aug-75	78	F

COMP	**45**
MCS	**69**
ICS	**72**
BIT	**62.5**

```
SELECT Course_ID, Add2, Avg(mark) AS AvgOfmark
FROM student
GROUP BY Course_ID;
```

Error message

Does this make sense?

Student

ID	Name	Add1	Add2	Pcode	Course_ID*	Year	DOB	Mark	Gender
1	Jones	10 Old Street	Bournemouth	BH11	ICS	3	03-Jun-82	66	M
3	Bloggs	12 Alder Way	Bournemouth	BH15	BIT	1	05-Sep-80	40	M
4	Johnson	11 Ashley Road	Poole	BH12	MCS	1	09-Mar-77	69	F
5	Walker	99 Oldcott Road	Bournemouth	BH44	ICS	2	15-May-82	78	M
11	Harrison	10 Daly Road	Bournemouth	BH7	BIT	2	24-Jul-70	85	F
12	Swift	6 Church St	Poole	BH9	COMP	1	30-Apr-72	12	M
13	Chambers	98 High St	Poole	BH5	COMP	2	06-Aug-75	78	F

Poole	**COMP**	**45**
Poole	**MCS**	**69**
Bournemouth	**ICS**	**72**
Bournemouth	**BIT**	**62.5**

```
SELECT Course_ID, Add2, Avg(mark) AS AvgOfmark
FROM student
GROUP BY Course_ID, Add2;
```

You can also use the 'Where' clause

```
SELECT deptno, MAX(sal)
FROM emp
WHERE job = 'MANAGER'
GROUP BY deptno;
```

```
DEPTNO         MAX(SAL)
------         --------
10             2450
20             2975
30             2850
```

Using function as selection criteria

```
SELECT job, AVG(sal)
  FROM emp
  GROUP BY job
  WHERE AVG(sal)>=3000;
```

Using function as selection criteria

WRONG!

```
       job, AVG(sal)

GROU         b
WHERE AVG(sal)>=3000;
```

Using function as selection criteria

WRONG!

```
        job, AVG(sal)

   GROU       b
   WHERE AVG(sal)>=3000;
```

```
SELECT job, AVG(sal)
   FROM emp
   GROUP BY job
   HAVING AVG(sal)>=3000;
```

```
JOB                 AVG(SAL)
------              --------
ANALYST             3000
PRESIDENT           5000
```