SQL 2

Unit 1.3

Logical Operators

- NOT, AND, OR (decreasing precedence), the usual operators on boolean values.

Fine those who are neither accountants nor analysts who are currently paid £16,000 to £30,000:

```
SELECT      empno
FROM        jobhistory
WHERE       salary BETWEEN 16000 AND 30000
AND         enddate IS NULL
AND         NOT (   position LIKE '%Accountant%' OR
                    position LIKE '%Analyst%' );
```

IN

- IN (list of values) determines whether a specified value is in a set of one or more listed values.

List the names of employees in departments 3 or 4 born before 1950:

```
SELECT        forenames,surname
FROM          employee
WHERE         depno IN (3,4)
  AND         enddate IS NULL
  AND         dob < '1-jan-1950';
```

Other SELECT capabilities

- SET or AGGREGATE functions
  - COUNT counts the rows in a table or group
  - SUM, AVERAGE, MIN, MAX - undertake the indicated operation on numeric columns of a table or group.
- GROUP BY - forms the result of a query into groups. Set functions can then be applied to these groups.
- HAVING - applies conditions to choose GROUPS of interest.

Simple COUNT examples

- How many employees are there?

```
SELECT          COUNT(*)
FROM            employee;
```

- What is the total salary bill?

```
SELECT          SUM(salary) totalsalary
FROM            jobhistory
WHERE           enddate IS NULL;
```

NOTE - the column title, ‘totalsalary’, to be printed with the result.

Grouped COUNTs

- How many employees are there in each department?

```
SELECT depno, COUNT(depno)
FROM employee
GROUP BY depno;
```

- How many employees are there in each department with more than 6 people?

```
SELECT depno, COUNT(depno)
FROM employee
GROUP BY depno
HAVING COUNT(depno) > 6;
```

The select lists above can include only set functions and the column(s) specified in the GROUP BY clause.

Joining Tables

- The information required to answer a query may be spread over two or more tables
- Tables specified in the FROM clause are conceptually joined so that each row in one table is combined with all rows in the other tables giving a new table, the Cartesian Product.
- A table with M rows combined with a table of N rows will produce a Cartesian Product of M × N rows;
- Information of interest is extracted from the Cartesian Product as specified by conditions in the WHERE clause.

Joining Tables cont…

- Join conditions in the WHERE clause equating foreign keys to primary keys eliminate invalid row combinations from the Cartesian Product.
- Join using the equality comparison operator are called Equi-joins.
- In general if there are N tables to be combined then (N-1) join conditions will be required (If there is a compound primary key with say two attributes one join condition will require two conditional statements).
- Further conditions may be included to obtain just those rows required to satisfy the current query.

Joining Tables cont…

List the names and current positions of employees in departments 3 or 4 who were born before 1950:

```
SELECT forenames,surname,position
FROM employee,jobhistory
WHERE employee.empno = jobhistory.empno -- Equi-join
AND depno IN (3,4)
AND dob < ’1-jan-1950’
AND enddate IS NULL;
```

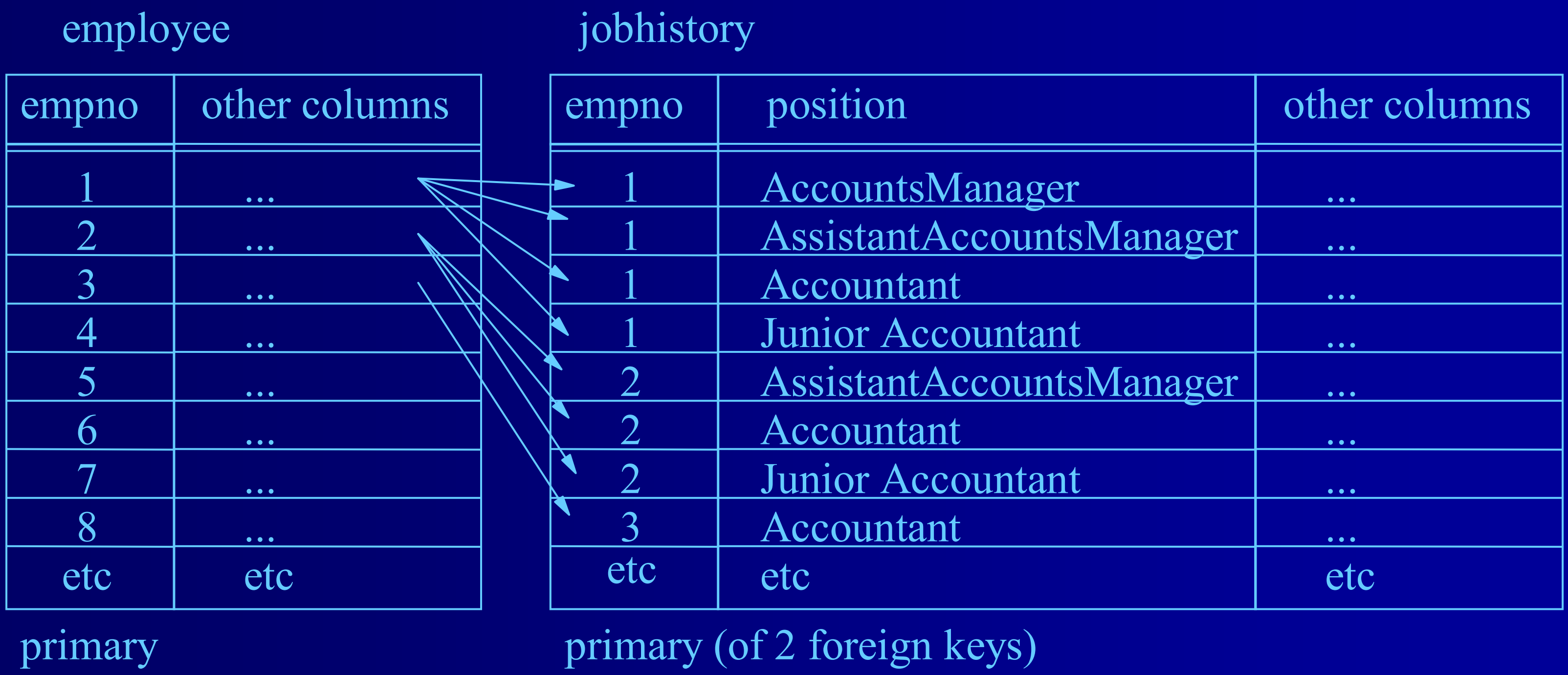
NOTE - the order of the WHERE predicates is not significant, and note the need to qualify empno with the table name.

SELECT - Order of Evaluation

SELECT [DISTINCT] column_name	**5,6** eliminate unwanted data
FROM label_list	**1** Cartesian Product
[WHERE condition]	**2** eliminate unwanted rows
[GROUP BY column_list	**3** group rows
[HAVING condition]]	**4** eliminate unwanted groups
[ORDER BY column_list[DESC]]	**7** sort rows

The last four components are optional.

One-to-Many Relationships



employee

empno	other columns
1	...
2	...
3	...
4	...
5	...
6	...
7	...
8	...
etc	etc

primary

jobhistory

empno	position	other columns
1	AccountsManager	...
1	AssistantAccountsManager	...
1	Accountant	...
1	Junior Accountant	...
2	AssistantAccountsManager	...
2	Accountant	...
2	Junior Accountant	...
3	Accountant	...
etc	etc	etc

primary (of 2 foreign keys)

Many-to-Many Relationships.

```
create table course (     courseno integer primary key,
                          cname varchar(20),
                          cdate date );
```

- Given the above course table, relationships between employees and courses can be represented by a table, commonly called a linker table, which implements many-to-many relationships
  - empno - foreign key references employer
  - course - foreign key references course

M-M Relationships cont…

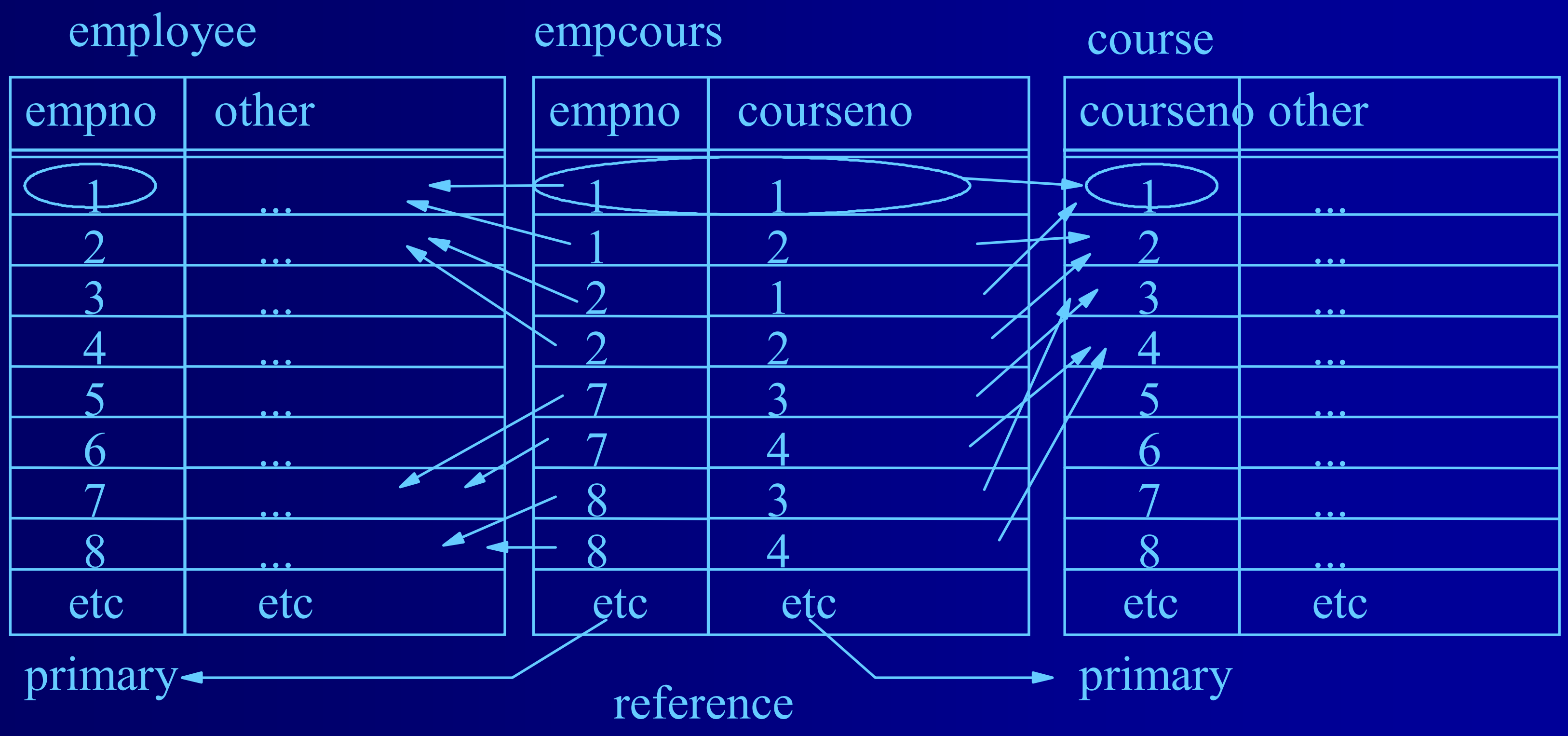
The ‘linker table’ that implements the many-to-many relationship:

```
create table empcourse
(
        empno              integer references employee (empno),
        courseno           integer references (courseno),
                  primary key (empno,courseno)
);
```

The **primary key** of empcourse is the **combination** (empno,course) and must be unique.

A linker table would commonly also hold information about the relationship. For the example above, the assessment of the employer on a particular course might usefully be included.

M-to-M cont…



employee

empno	other
1	…
2	…
3	…
4	…
5	…
6	…
7	…
8	…
etc	etc

empcours

empno	courseno
1	1
1	2
2	1
2	2
7	3
7	4
8	3
8	4
etc	etc

course

courseno	other
1	…
2	…
3	…
4	…
5	…
6	…
7	…
8	…
etc	etc

primary

reference

primary

Aliases

- Temporary labels, aliases can be defined for table names in the FROM clause and can then be used wherever the table name might appear.
- Aliases may simply be introduced as a shorthand, or to add clarity to select statements.

List employee numbers with surname and current job title:

```
SELECT      emp.empno, emp.surname, jh.position
FROM        employee emp, jobhistory jh
WHERE       emp.empno = jh.empno
   AND      jh.enddate IS NULL;
```

Aliases with Self Joins

- In the previous example the aliases were cosmetic but they become essential if one table is incorporated two or more times into one enquiry, as in Self Joins which occur when one table is joined to itself.

Name employees younger than Liza Brunell:

```
SELECT      young.surname, young.forenames
FROM        employee young, employee liza
WHERE       liza.forenames = ’Liza’
   AND      liza.surname = ’Brunell’
   AND      young.dob > liza.dob;
```