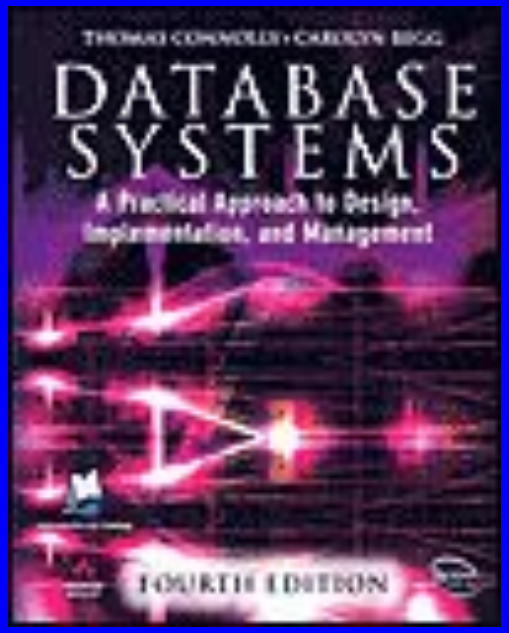
Lecture Five

SQL – Data Manipulation

Based on Chapter Five of this book:



Database Systems: A Practical Approach to Design, Implementation and Management

International Computer Science S.

Carolyn Begg, Thomas Connolly

Lecture - Objectives

- Purpose and importance of SQL.
- How to retrieve data from database using SELECT and:
  - Use compound WHERE conditions.
  - Sort query results using ORDER BY.
- How to update database using INSERT, UPDATE, and DELETE.

Objectives of SQL

- Ideally, database language should allow user to:
  - create the database and relation structures;
  - perform insertion, modification, deletion of data from relations;
  - perform simple and complex queries.
- Must perform these tasks with minimal user effort and command structure/syntax must be easy to learn.
- It must be portable.

Objectives of SQL

- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.
- Until SQL3, SQL did not contain flow of control commands. These had to be implemented using a programming or job-control language, or interactively by the decisions of user.

Objectives of SQL

- **SQL is relatively easy to learn:**
  - **it is non-procedural - you specify *what* information you require, rather than *how* to get it;**

Objectives of SQL

- Consists of standard English words:

```
CREATE TABLE Staff(staffNo VARCHAR(5),
                   lName VARCHAR(15),
                   salary DECIMAL(7,2));
INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
SELECT staffNo, lName, salary
    FROM Staff
    WHERE salary > 10000;
```

Objectives of SQL

- **Can be used by range of users including DBAs, management, application developers, and other types of end users.**
- **An ISO standard now exists for SQL, making it both the formal and *de facto* standard language for relational databases.**

Writing SQL Commands

- **SQL statement consists of *reserved words* and *user-defined words*.**
  - **Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.**
  - **User-defined words are made up by user and represent names of various database objects such as relations, columns, views.**

Writing SQL Commands

- **Most components of an SQL statement are *case insensitive*, except for literal character data.**
- **More readable with indentation and lineation:**

  - Each clause should begin on a new line.

  - Start of a clause should line up with start of other clauses.

  - If clause has several parts, should each appear on a separate line and be indented under start of clause.

Writing SQL Commands

- **Use extended form of BNF notation:**
  - **Upper-case letters represent reserved words.**
  - **Lower-case letters represent user-defined words.**
  - **| indicates a *choice* among alternatives.**
  - **Curly braces indicate a *required element*.**
  - **Square brackets indicate an *optional element*.**
  - **… indicates *optional repetition* (0 or more).**

Literals

- Literals are constants used in SQL statements.
- All non-numeric literals must be enclosed in single quotes (e.g. ‘London’).
- All numeric literals must not be enclosed in quotes (e.g. 650.00).

SELECT Statement

```
SELECT [DISTINCT | ALL]
   {* | [columnExpression [AS newName]] [,...] }
FROM       TableName [alias] [, ...]
[WHERE    condition]
[GROUP BY    columnList]  [HAVING condition]
[ORDER BY    columnList]
```

SELECT Statement

FROM	Specifies table(s) to be used.
WHERE	Filters rows.
GROUP BY	Forms groups of rows with same column value.
HAVING	Filters groups subject to some condition.
SELECT	Specifies which columns are to appear in output.
ORDER BY	Specifies the order of the output.

SELECT Statement

- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.

Example 5.1 All Columns, All Rows

List full details of all staff.

```
SELECT staffNo, fName, lName, address,
       position, sex, DOB, salary, branchNo
  FROM Staff;
```

- Can use * as an abbreviation for 'all columns':

```
SELECT *
FROM Staff;
```

Example 5.1 All Columns, All Rows

Table 5.1 Result table for Example 5.1.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000.00	B005

Example 5.2 Specific Columns, All Rows

Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary
FROM Staff;
```

Example 5.2 Specific Columns, All Rows

Table 5.2 Result table for Example 5.2.

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Example 5.3 Use of DISTINCT

List the property numbers of all properties that have been viewed.

```
SELECT propertyNo
    FROM Viewing;
```

propertyNo
PA14
PG4
PG4
PA14
PG36

Example 5.3 Use of DISTINCT

- Use DISTINCT to eliminate duplicates:

```
SELECT DISTINCT propertyNo
    FROM Viewing;
```

propertyNo
PA14
PG4
PG36

Example Use of DISTINCT

- Produce a list showing the different types of property available in each city

```
SELECT city, type
    FROM PropertyForRent;
```

city	type
Aberdeen	House
London	Flat
Glasgow	Flat
Glasgow	Flat
Glasgow	House
Glasgow	Flat

Example Use of DISTINCT

- Use DISTINCT to eliminate duplicates

```
SELECT DISTINCT city, type
FROM PropertyForRent;
```

city	type
Aberdeen	House
London	Flat
Glasgow	Flat
Glasgow	House

Example 5.4 Calculated Fields

Produce a list of monthly salaries for all staff, showing staff number, first and last names, and salary details.

```
SELECT staffNo, fName, lName, salary/12
    FROM Staff;
```

Table 5.4 Result table for Example 5.4.

staffNo	fName	lName	col4
SL21	John	White	2500.00
SG37	Ann	Beech	1000.00
SG14	David	Ford	1500.00
SA9	Mary	Howe	750.00
SG5	Susan	Brand	2000.00
SL41	Julie	Lee	750.00

Example 5.5 Comparison Search Condition

List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 10000;
```

Table 5.5 Result table for Example 5.5.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Example 5.6 Compound Comparison Search Condition

List addresses of all branch offices in London or Glasgow.

```
SELECT *
    FROM Branch
    WHERE city = 'London' OR city = 'Glasgow';
```

Table 5.6 Result table for Example 5.6.

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B003	163 Main St	Glasgow	G11 9QX
B002	56 Clover Dr	London	NW10 6EU

Example 5.7 Range Search Condition

List all staff with a salary between 20,000 and 30,000.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary BETWEEN 20000 AND 30000;
```

- BETWEEN test includes the endpoints of range.

Example 5.7 Range Search Condition

Table 5.7 Result table for Example 5.7.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG5	Susan	Brand	Manager	24000.00

Example 5.7 Range Search Condition

- Also a negated version NOT BETWEEN.
- BETWEEN does not add much to SQL's expressive power Could also write:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary>=20000 AND salary <= 30000;
```

- Useful, though, for a range of values.

Example 5.8 Set Membership

List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position IN ('Manager', 'Supervisor');
```

Table 5.8 Result table for Example 5.8.

staffNo	fName	lName	position
SL21	John	White	Manager
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Example 5.8 Set Membership

- There is a negated version (NOT IN).
- IN does not add much to SQL's expressive power.
- Could have expressed this as:

```
SELECT staffNo, fName, lName, position
 FROM Staff
WHERE position='Manager' OR
            position='Supervisor';
```

- IN is more efficient when set contains many values.

Example 5.9 Pattern Matching

Find all owners with the string 'Glasgow' in their address.

```
SELECT ownerNo, fName, lName, address, telNo
FROM PrivateOwner
WHERE address LIKE '%Glasgow%';
```

Table 5.9 Result table for Example 5.9.

ownerNo	fName	lName	address	telNo
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025

Example 5.9 Pattern Matching

- SQL has two special pattern matching symbols:
  - %: sequence of zero or more characters;
  - _ (underscore): any single character.
- LIKE '%Glasgow%' means a sequence of characters of any length containing '*Glasgow*'.

Example 5.10 NULL Search Condition

List details of all viewings on property PG4 where a comment has not been supplied.

- There are 2 viewings for property PG4, one with and one without a comment.
- Have to test for null explicitly using special keyword IS NULL:

```
SELECT clientNo, viewDate
    FROM Viewing
    WHERE propertyNo = 'PG4' AND
                comment IS NULL;
```

Example 5.10 NULL Search Condition

Table 5.10 Result table for Example 5.10.

clientNo	viewDate
CR56	26-May-01

- **Negated version (IS NOT NULL) can test for non-null values.**

Example 5.11 Single Column Ordering

List salaries for all staff, arranged in descending order of salary.

```
SELECT staffNo, fName, lName, salary
    FROM Staff
    ORDER BY salary DESC;
```

Example 5.11 Single Column Ordering

Table 5.11 Result table for Example 5.11.

staffNo	fName	lName	salary
SL21	John	White	30000.00
SG5	Susan	Brand	24000.00
SG14	David	Ford	18000.00
SG37	Ann	Beech	12000.00
SA9	Mary	Howe	9000.00
SL41	Julie	Lee	9000.00

Example 5.12 Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

```
SELECT propertyNo, type, rooms, rent
    FROM PropertyForRent
    ORDER BY type;
```

Example 5.12 Multiple Column Ordering

Table 5.12(a) Result table for Example 5.12 with one sort key.

propertyNo	type	rooms	rent
PL94	Flat	4	400
PG4	Flat	3	350
PG36	Flat	3	375
PG16	Flat	4	450
PA14	House	6	650
PG21	House	5	600

Example 5.12 Multiple Column Ordering

- Four flats in this list - as no minor sort key specified, system arranges these rows in any order it chooses.
- To arrange in order of rent, specify minor order:

```
SELECT propertyNo, type, rooms, rent
    FROM PropertyForRent
    ORDER BY type, rent DESC;
```

Example 5.12 Multiple Column Ordering

Table 5.12(b) Result table for Example 5.12 with two sort keys.

propertyNo	type	rooms	rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	375
PG4	Flat	3	350
PA14	House	6	650
PG21	House	5	600

INSERT

INSERT INTO TableName (columnList)
VALUES (dataValueList)

- **Any columns omitted from "columnList" must have been declared as NULL when table was created, unless a DEFAULT was specified when creating the column.**

INSERT

- *dataValueList* must match *columnList* as follows:
  - number of items in each list must be same;
  - must be direct correspondence in position of items in two lists;
  - data type of each item in *dataValueList* must be compatible with data type of corresponding column.

Example 5.35 INSERT … VALUES

Insert a new row into Staff table supplying data for all columns.

```
INSERT INTO Staff (staffNo, fName, lName,
position, sex, DOB, salary, branchNo)
VALUES ('SG16', 'Alan', 'Brown',        'Assistant',
   'M', '1957-05-25', 8300, 'B003');
```

Example 5.36 INSERT using Defaults

Insert a new row into Staff table supplying data for all mandatory columns.

```
INSERT INTO Staff (staffNo, fName, lName,
                   position, salary, branchNo)
 VALUES ('SG44', 'Anne', 'Jones',
         'Assistant', 8100, 'B003');
```

UPDATE

```
UPDATE TableName
SET columnName1 = dataValue1
    [, columnName2 = dataValue2...]
[WHERE searchCondition]
```

- *TableName* can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.

UPDATE

- **WHERE clause is optional:**
  - **if omitted, named columns are updated for all rows in table.;**
  - **if specified, only those rows that satisfy *searchCondition* are updated.**
- **New *dataValue(s)* must be compatible with data type for corresponding column.**

Example 5.38/39 UPDATE All Rows

Give all staff a 3% pay increase.

```
UPDATE Staff
SET salary = salary*1.03;
```

Give all Managers a 5% pay increase.

```
UPDATE Staff
    SET salary = salary*1.05
    WHERE position = 'Manager';
```

Example 5.40 UPDATE Multiple Columns

Promote David Ford (staffNo = 'SG14') to Manager and change his salary to 18,000.

```
UPDATE Staff
    SET position = 'Manager', salary = 18000
    WHERE staffNo = 'SG14';
```

DELETE

DELETE FROM TableName
[WHERE searchCondition]

- ***TableName*** **can be name of a base table or an updatable view.**
- ***searchCondition*** **is optional; if omitted, all rows are deleted from table. This does not delete table. If *search_condition* is specified, only those rows that satisfy condition are deleted.**

Example 5.41/42 DELETE Specific Rows

Delete all viewings that relate to property PG4.

```
DELETE FROM Viewing
    WHERE propertyNo = 'PG4';
```

Delete all records from the Viewing table.

```
DELETE FROM Viewing;
```

Lecture – Further Objectives

- How to retrieve data from database using SELECT and:
  - Use aggregate functions.
  - Group data using GROUP BY and HAVING.
  - Use subqueries.
  - Join tables together.
  - Perform set operations (UNION, INTERSECT, EXCEPT).

SELECT Statement - Aggregates

- ISO standard defines five aggregate functions:

COUNT returns number of values in specified column.

SUM returns sum of values in specified column.

AVG returns average of values in specified column.

MIN returns smallest value in specified column.

MAX returns largest value in specified column.

SELECT Statement - Aggregates

- Each operates on a single column of a table and return single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric columns, but SUM and AVG may be used on numeric columns only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.

SELECT Statement - Aggregates

- COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.
- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.

SELECT Statement - Aggregates

- Aggregate functions can be used only in SELECT list and in HAVING clause.
- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column outwith an aggregate function. For example, following is illegal:

```
SELECT staffNo, COUNT(salary)
FROM Staff;
```

Example 5.13 Use of COUNT(*)

How many properties cost more than 350 per month to rent?

```
SELECT COUNT(*) AS count
    FROM PropertyForRent
    WHERE rent > 350;
```

Table 5.13 Result table for Example 5.13.

count
5

Example 5.14 Use of COUNT(DISTINCT)

How many different properties viewed in May '01?

```
SELECT COUNT(DISTINCT propertyNo) AS count
  FROM Viewing
  WHERE date BETWEEN '1-May-01'
        AND '31-May-01';
```

Table 5.14 Result table for Example 5.14.

count
2

Example 5.15 Use of COUNT and SUM

Find number of Managers and sum of their salaries.

```
SELECT COUNT(staffNo) AS count,
       SUM(salary) AS sum
FROM Staff
WHERE position = 'Manager';
```

Table 5.15 Result table for Example 5.15.

count	sum
2	54000.00

Example 5.16 Use of MIN, MAX, AVG

Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS min,
  MAX(salary) AS max,
  AVG(salary) AS avg
FROM Staff;
```

Table 5.16 Result table for Example 5.16.

min	max	avg
9000.00	30000.00	17000.00

SELECT Statement - Grouping

- Use GROUP BY clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be *single-valued per group*, and SELECT clause may only contain:
  - Column names
  - Aggregate functions
  - Constants
  - Expression involving combinations of the above.

SELECT Statement - Grouping

- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

Example 5.17 Use of GROUP BY

Find number of staff in each branch and their total salaries.

```
SELECT    branchNo,
                COUNT(staffNo) AS count,
          SUM(salary) AS sum
FROM Staff
GROUP BY branchNo
ORDER BY branchNo;
```

Example 5.17 Use of GROUP BY

Table 5.17 Result table for Example 5.17.

branchNo	count	sum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

Example 5.17 Use of GROUP BY

Find number of assistants in each branch and the total salary bill for all assistants

```
SELECT    branchNo,
                COUNT(staffNo) AS count,
          SUM(salary) AS sum
FROM Staff
WHERE position = 'Assistant'
GROUP BY branchNo
ORDER BY branchNo;
```

Example 5.17 Use of GROUP BY

branchNo	count	sum
B003	1	12000.00
B005	1	9000.00
B007	1	9000.00

Restricted Groupings – HAVING clause

- HAVING clause is designed for use with GROUP BY to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

Example 5.18 Use of HAVING

For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT branchNo,
            COUNT(staffNo) AS count,
       SUM(salary) AS sum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

Example 5.18 Use of HAVING

Table 5.18 Result table for Example 5.18.

branchNo	count	sum
B003	3	54000.00
B005	2	39000.00

Subqueries

- **Some SQL statements can have a SELECT embedded within them.**
- **A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a *subquery* or *nested query*.**
- **Subselects may also appear in INSERT, UPDATE, and DELETEs.**

Example 5.19 Subquery with Equality

List staff who work in branch at '163 Main St'.

```
SELECT staffNo, fName, lName, position
    FROM Staff
    WHERE branchNo =
        (SELECT branchNo
         FROM Branch
         WHERE street = '163 Main St');
```

Example 5.19 Subquery with Equality

- Inner SELECT finds branch number for branch at '163 Main St' ('B003').
- Outer SELECT then retrieves details of all staff who work at this branch.
- Outer SELECT then becomes:

```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE branchNo = 'B003';
```

Example 5.19 Subquery with Equality

Table 5.19 Result table for Example 5.19.

staffNo	fName	lName	position
SG37	Ann	Beech	Assistant
SG14	David	Ford	Supervisor
SG5	Susan	Brand	Manager

Example 5.20 Subquery with Aggregate

List all staff whose salary is greater than the average salary.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary >
        (SELECT AVG(salary)
         FROM Staff);
```

Example 5.20 Subquery with Aggregate

- Cannot write 'WHERE salary > AVG(salary)'
- Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > 17000;
```

Example 5.20 Subquery with Aggregate

Table 5.20 Result table for Example 5.20.

staffNo	fName	lName	position	salDiff
SL21	John	White	Manager	13000.00
SG14	David	Ford	Supervisor	1000.00
SG5	Susan	Brand	Manager	7000.00

Subquery Rules

- **ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).**
- **Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.**
- **By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.**

Subquery Rules

- When subquery is an operand in a comparison, subquery must appear on right-hand side.
- A subquery may not be used as an operand in an expression.

Multi-Table Queries

- Can use subqueries provided result columns come from same table.
- If result columns come from more than one table must use a join.
- To perform join, include more than one table in FROM clause.
- Use comma as separator and typically include WHERE clause to specify join column(s).

Multi-Table Queries

- Also possible to use an alias for a table named in FROM clause.
- Alias is separated from table name with a space.
- Alias can be used to qualify column names when there is ambiguity.

Example 5.24 Simple Join

List names of all clients who have viewed a property along with any comment supplied.

```
SELECT c.clientNo, fName, lName,
                propertyNo, comment
FROM Client c, Viewing v
WHERE c.clientNo = v.clientNo;
```

Example 5.24 Simple Join

- Only those rows from both tables that have identical values in the clientNo columns (c.clientNo = v.clientNo) are included in result.
- Equivalent to equi-join in relational algebra.

Table 5.24 Result table for Example 5.24.

clientNo	fName	lName	propertyNo	comment
CR56	Aline	Stewart	PG36	
CR56	Aline	Stewart	PA14	too small
CR56	Aline	Stewart	PG4	
CR62	Mary	Tregear	PA14	no dining room
CR76	John	Kay	PG4	too remote

Alternative JOIN Constructs

- SQL provides alternative ways to specify joins:

  FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo
  FROM Client JOIN Viewing USING clientNo
  FROM Client NATURAL JOIN Viewing

- In each case, FROM replaces original FROM and WHERE. However, first produces table with two identical clientNo columns.

Example 5.25 Sorting a join

For each branch, list numbers and names of staff who manage properties, and properties they manage.

```
SELECT s.branchNo, s.staffNo, fName, lName,
                propertyNo
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
ORDER BY s.branchNo, s.staffNo, propertyNo;
```

Example 5.25 Sorting a join

Table 5.25 Result table for Example 5.25.

branchNo	staffNo	fName	lName	propertyNo
B003	SG14	David	Ford	PG16
B003	SG37	Ann	Beech	PG21
B003	SG37	Ann	Beech	PG36
B005	SL41	Julie	Lee	PL94
B007	SA9	Mary	Howe	PA14

Example 5.26 Three Table Join

For each branch, list staff who manage properties, including city in which branch is located and properties they manage.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName,
       propertyNo
FROM branch b, staff s, property_for_rent p
WHERE b.branchNo = s.branchNo AND
       s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

Example 5.26 Three Table Join

Table 5.26 Result table for Example 5.26.

branchNo	city	staffNo	fName	lName	propertyNo
B003	Glasgow	SG14	David	Ford	PG16
B003	Glasgow	SG37	Ann	Beech	PG21
B003	Glasgow	SG37	Ann	Beech	PG36
B005	London	SL41	Julie	Lee	PL94
B007	Aberdeen	SA9	Mary	Howe	PA14

- **Alternative formulation for FROM and WHERE:**

FROM (branch b JOIN Staff s USING branchNo) AS bs JOIN PropertyForRent p USING staffNo

Example 5.27 Multiple Grouping Columns

Find number of properties handled by each staff member.

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS count
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo
GROUP BY s.branchNo, s.staffNo
ORDER BY s.branchNo, s.staffNo;
```

Example 5.27 Multiple Grouping Columns

Table 5.27(a) Result table for Example 5.27.

branchNo	staffNo	count
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Computing a Join

Procedure for generating results of a join are:

1. Form Cartesian product of the tables named in FROM clause.
2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.
3. For each remaining row, determine value of each item in SELECT list to produce a single row in result table.

Computing a Join

4. If DISTINCT has been specified, eliminate any duplicate rows from the result table.

5. If there is an ORDER BY clause, sort result table as required.

- SQL provides special format of SELECT for Cartesian product:

```
SELECT [DISTINCT | ALL] {* | columnList}
FROM Table1 CROSS JOIN Table2
```

Outer Joins

- If one row of a joined table is unmatched, row is omitted from result table.
- Outer join operations retain rows that do not satisfy the join condition.
- Consider following tables:

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Outer Joins

- The (inner) join of these two tables:

```
SELECT branchNo, bCity, propertyNo, pCity
FROM Branch1, PropertyForRent1
WHERE bCity = pCity;
```

Table 5.27(b) Result table for inner join of Branch1 and PropertyForRent1 tables.

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Outer Joins

- Result table has two rows where cities are same.
- There are no rows corresponding to branches in Bristol and Aberdeen.
- To include unmatched rows in result table, use an Outer join.

Example 5.28 Left Outer Join

List branches and properties that are in same city along with any unmatched branches.

```
SELECT branchNo, bCity, propertyNo, pCity
FROM Branch1 LEFT JOIN PropertyForRent1
                    ON bCity = pCity;
```

```
SELECT branchNo, bCity, propertyNo, pCity
FROM Branch1,  PropertyForRent1
WHERE bCity = pCity (+);
```

Example 5.28 Left Outer Join

- Includes those rows of first (left) table unmatched with rows from second (right) table.
- Columns from second table are filled with NULLs.

Table 5.28 Result table for Example 5.28.

branchNo	bCity	propertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

Example 5.29 Right Outer Join

List branches and properties in same city and any unmatched properties.

```
SELECT branchNo, bCity, propertyNo, pCity
FROM Branch1 RIGHT JOIN PropertyForRent1
                ON bCity = pCity;
```

```
SELECT branchNo, bCity, propertyNo, pCity
FROM Branch1,  PropertyForRent1
WHERE bCity (+) = pCity;
```

Example 5.29 Right Outer Join

- Right Outer join includes those rows of second (right) table that are unmatched with rows from first (left) table.
- Columns from first table are filled with NULLs.

Table 5.29 Result table for Example 5.29.

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

Example 5.30 Full Outer Join

List branches and properties in same city and any unmatched branches or properties.

```
SELECT branchNo, bCity, propertyNo, pCity
FROM Branch1 FULL JOIN PropertyForRent1
                ON bCity = pCity;
```

```
SELECT branchNo, bCity, propertyNo, pCity
FROM Branch1,  PropertyForRent1
WHERE bCity (+) = pCity (+)
```

Example 5.30 Full Outer Join

- Includes rows that are unmatched in both tables.
- Unmatched columns are filled with NULLs.

Table 5.30 Result table for Example 5.30.

branchNo	bCity	propertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS are for use only with subqueries.
- Produce a simple true/false result.
- True if and only if there exists at least one row in result table returned by subquery.
- False if subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.

EXISTS and NOT EXISTS

- As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.
- Common for subqueries following (NOT) EXISTS to be of form:

  (SELECT * ...)

Example 5.31 Query using EXISTS

Find all staff who work in a London branch.

```
SELECT staffNo, fName, lName, position
FROM Staff s
WHERE EXISTS
   (SELECT *
    FROM Branch b
    WHERE s.branchNo = b.branchNo AND
          city = 'London');
```

Example 5.31 Query using EXISTS

Table 5.31 Result table for Example 5.31.

staffNo	fName	lName	position
SL21	John	White	Manager
SL41	Julie	Lee	Assistant

Example 5.31 Query using EXISTS

- Note, search condition s.branchNo = b.branchNo is necessary to consider correct branch record for each member of staff.
- If omitted, would get all staff records listed out because subquery:

```
SELECT * FROM Branch WHERE city='London'
```

- would be always be true and query would be:

```
SELECT staffNo, fName, lName, position FROM Staff
WHERE true;
```

Example 5.31 Query using EXISTS

- Could also write this query using join construct:

```
SELECT staffNo, fName, lName, position
FROM Staff s, Branch b
WHERE s.branchNo = b.branchNo
AND city = 'London';
```