Chapter 6

DML

Data Manipulation Language

Adapted from

Database Systems, Thomas Connoll, Carolyn Begg, 6th Eds

Master Slides downloaded from : *https://cap261.files.wordpress.com/2011/01/chapter5.ppt*

Relational ALgebra

Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{(\text{<join attributes 1>}), (\text{<join attributes 2>})} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{(\text{<join attributes 1>}), (\text{<join attributes 2>})} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Relational Algebra

Definition

- Relational Algebra is Query Language
  - Collection of high level operators that operate on relations.
  - Theoretical, Procedural Language
  - Purpose is data manipulation
  - Method is to write expressions
  - Six Fundamental Operators
    - select (unary)
    - project (unary)
    - rename (unary)
    - cartesian product (binary)
    - union (binary)
    - set-difference (binary)
  - Other operators defined in terms of fundamental operators
- SQL can be mapped into relational algebra operations

Relational Algebra

Example

Given:

1. Animal (Animal_name, food, nlegs)
2. Keeper(keeper#, keeper_name)
3. Supervision(keeper#, animal_name)

Queries:

1. What does a camel eat?
   - (PROJECT, RESTRICT)
2. What is supervised by a keeper called Morris?
   - (JOIN, RESTRICT, PROJECT)

Relational Algebra

Example

Given:

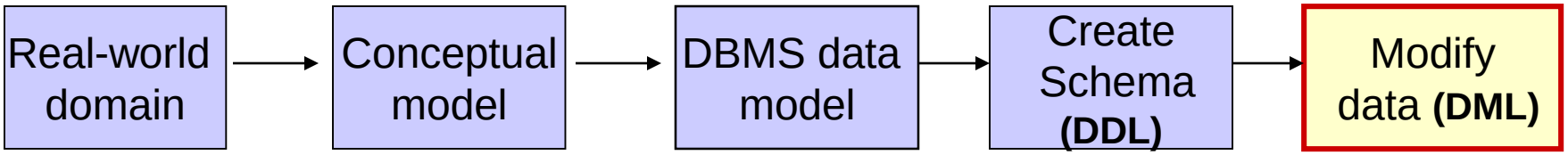
1. Book (ISBN, Price, Title)
2. Author(AuthorID, AuthorName)
3. Book/Author(AuthorID, ISBN)

Queries:

1. What is the Price of the book “War and Peace”?
   - (PROJECT, RESTRICT)
2. Who is the author of the book War and Peace?
   - (JOIN, RESTRICT, PROJECT)
3. Find all the books written by author Shakespeare?
   - (JOIN, RESTRICT, PROJECT)

Database Design

Steps in building a database for an application:



Data Manipulation Language (DML) Statements

The main SQL data manipulation language statements are:

SELECT

INSERT INTO

UPDATE

DELETE FROM

Notations

Notations to define SQL statements:

- UPPER-CASE letters represents reserved words.
- Lower-case letters represents user-defined words.
- **|** indicates a choice among alternatives; (e.g. a | b | c).
- { } indicates a **required** element.
- [ ] indicates an **optional** element.
- … indicates **optional** repetition of an item zero or more times.
- Underlined words represent default values.

SELECT

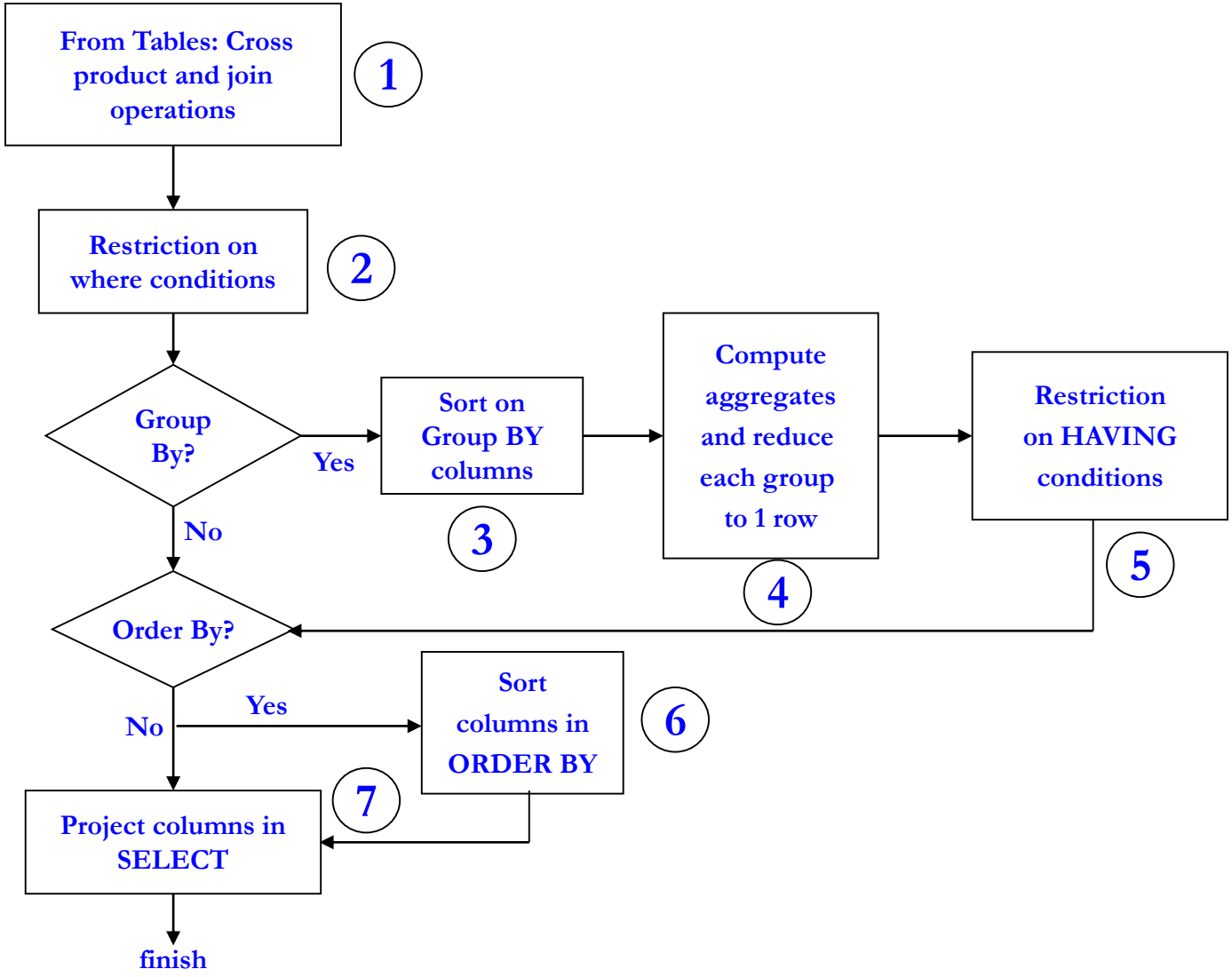
Syntax

```
SELECT [DISTINCT|ALL]{*|column|column_expression [AS new_name][,…]}
   FROM   table_name [alias] [, … ]
     [WHERE  condition]
     [GROUP BY  column_list]
     [HAVING  condition]
     [ORDER BY  column_list [ASC|DESC]];
```

- *column* represents a column name.
- *column_expression* represents an expression on a column.
- *table_name* is the name of an existing database table or view.
- FROM specifies the table(s) to be used.
- WHERE filters the rows subject to some condition.
- GROUP BY forms groups of rows with the same column name.
- SELECT specifies which column are to appear in the output.
- ORDER BY specifies the order of the output.
- Order of the clauses in the SELECT statement can not be changed.
- The result of a query is another table.
- Asterisk (*) means all columns.

Select Clause

Conceptual Evaluation



Retrieve all columns & rows

Syntax

```
SELECT {* | column| column_expression [,…]}
   FROM   table_name;
```

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Retrieve all staff information.

```
SELECT  sno, fname, lname, position, sex, dob, salary, bno
  FROM  staff;
```

OR

```
SELECT  *
  FROM  staff;
```

Sno	FName	LName	position	Sex	DOB	Salary	bno
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000	B007
SG5	Susan	Brand	Manager	F	13-Jun-40	24000	B003

Retrieve specific columns & all rows

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List salaries of all staff, showing only the staff number, the first and last name, and salary.

```
SELECT  sno, fname, lname, salary
  FROM  staff;
```

Sno	FName	LName	Salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000

Use of DISTINCT

DISTINCT eliminates duplicated tuples.

Syntax

```
SELECT [DISTINCT|ALL] {* | column |column_expression [,…]}
   FROM   table_name;
```

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List the available positions for staff .

```
SELECT DISTINCT position
 FROM  staff;
```

position
Manager
Assistant
Supervisor
Assistant
Manager

```
SELECT position
 FROM  staff;
```

position
Manager
Assistant
Supervisor

```
SELECT DISTINCT position
 FROM  staff;
```

Calculated fields

- The SQL expression in the SELECT list specifies a derived field.
- Columns referenced in the arithmetic expression must have a numeric type.
- SQL expression can involve + , - , * , / , (,).
- AS clause is used to name the derived column.

Syntax

```
SELECT {* | column| column_expression [AS new_name] [,…]}
   FROM   table_name;
```

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List the monthly salaries for all staff, showing the staff number, the first and last names.

```
SELECT sno, fname, lname, salary/12 AS MonthlySalary
 FROM  staff;
```

Sno	FName	LName	MonthlySalary
SL21	John	White	2500
SG37	Ann	Beech	1000
SG14	David	Ford	1500
SA9	Mary	Howe	750
SG5	Susan	Brand	2000

Select Clause

Operators

- Arithmetic operators supported by SQL
  - () Parentheses
  - / Division
  - * Multiplication
  - - Subtraction
  - + Addition
- Associativity and Precedence:
  - Precedence is the order in which operators are evaluated
  - Associativity is the order in which operators of same precedence are evaluated
  - Multiplication and Division have the same precedence and Subtraction and Division have the same precedence.
  - Equal precedence operators are evaluated from right to left
  - Parentheses can be used to control the sequence of evaluation of various operators

Select Clause

Alias (as)

- Used to assign names to the columns when they are retrieved from the database table.
- **Syntax**:

```
Select expr1 [as alias1], expr2 [as alias2] [, …   ]
From table1 [, table2, …]
[Where condition]
```

FROM Clause

Alias

- Used to assign names to a table.
- **Syntax**:

  Select expr1 [as alias1], expr2 [as alias2] [, …]
  From table1 [T1, table2 [T2], …]
  [Where condition]

Row selection (WHERE clause)

WHERE clause consists of five basic search conditions:

- **Comparison:** Compare the value of one expression to the value of another expression (= , <, >, <=, >=, <>).
- **Range:** Test whether the value of an expression falls within a specified range of values (BETWEEN/ NOT BETWEEN).
- **Set membership:** Test whether the value of an expression equals one of a set of values (IN/ NOT IN).
- **Pattern match:** Test whether a string matches a specified pattern (LIKE/ NOT LIKE).
- **NULL:** Test whether a column has null value (IS NULL/ IS NOT NULL).

Simple Queries
Comparison search condition

Comparison operators: = , < , > , <= , >= , <>

Syntax

```
SELECT [DISTINCT|ALL] {* | column| [column_expression [AS
new_name]] [,…]}
   FROM   table_name
     [WHERE  condition];
```

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List all staff with a salary greater than 10,000. showing number, name and salary.

```
SELECT sno, fname, lname, salary
  FROM staff
    WHERE  salary > 10000;
```

Sno	FName	LName	Salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SG5	Susan	Brand	24000

Compound comparison search condition

Compound comparison operators: AND , OR , NOT , ( )

Order of evaluation:

- Expression is evaluated left to right
- Between brackets
- NOT
- AND
- OR

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List all staff who works as managers or assistants.

```
SELECT sno, fname, lname, position
 FROM staff
  WHERE  position = 'Manager' OR position = 'Assistant';
```

Sno	FName	LName	position
SL21	John	White	Manager
SG37	Ann	Beech	Assistant
SA9	Mary	Howe	Assistant
SG5	Susan	Brand	Manager

BETWEEN/ NOT BETWEEN

BETWEEN checks if a value is within a range.
NOT BETWEEN checks if a value is outside a range.

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List all staff with a salary between 20000 and 30000.

```
SELECT sno, fname, lname, salary
  FROM staff
    WHERE  salary BETWEEN 20000 AND 30000;
```

This would be expressed as:

```
SELECT sno, fname, lname, salary
  FROM staff
    WHERE  salary >= 20000 AND salary <= 30000;
```

Sno	FName	LName	Salary
SL21	John	White	30000
SG5	Susan	Brand	24000

IN/ NOT IN

IN tests whether a data value matches one of a list values.
NOT IN checks for data values that do not lie in a specific list of values.

Example: STAFF(sno, fname, lname, position, sex, dob, salary, bno)

List all Managers or Assistants.

```
SELECT sno, fname, lname, position
   FROM  staff
      WHERE   position IN ('Manager', 'Assistant');
```

This would be expressed as:

```
SELECT sno, fname, lname, position
  FROM  staff
   WHERE   position = 'Manager' OR position = 'Assistant';
```

Sno	FName	LName	position
SL21	John	White	Manager
SG37	Ann	Beech	Assistant
SA9	Mary	Howe	Assistant
SG5	Susan	Brand	Manager

LIKE/ NOT LIKE

SQL has special pattern matching symbol:

% represents any sequence of zero or more character (wildcard)

_ represents any single character

Example:

- Address LIKE ‘H%’ means that the first character must be *H*, but the rest can be anything.
- Address LIKE ‘H_ _ _’ means that there must be exactly four characters in the string, the first of which must be *H.*
- Address LIKE ‘%e’ means any sequence of characters, of length at least 1, with the last character an *e.*
- Address LIKE ‘%Glasgow%’ means a sequence of characters of any length containing *Glasgow.*
- Address NOT LIKE ‘H%’ means the first character can not be *H.*

LIKE/ NOT LIKE

If the search string can include the pattern-matching character itself, we can use an **escape** character to represent the pattern matching character.

'15%' is represented by LIKE '15#%' ESCAPE '#'

Example: STAFF(sno, fname, lname, position, sex, dob, salary, address, bno)

List all staff with the string 'Glasgow' in their address.

```
SELECT sno, fname, lname, address
   FROM  staff
      WHERE   address LIKE '%Glasgow%';
```

Sno	FName	LName	address
SL21	John	White	Achray St,Glasgow G32 9DX
SG37	Ann	Beech	Well St, Glasgow G42

IS NULL/ IS NOT NULL

NULL represents missing or unknown value.

NULL can does not represent a zero or a string of blank spaces.

A NULL value can not be tested with = or<> to another string.

We have to test for NULL explicitly.

Example:

VIEWING (ClientNo, PropertyNo, ViewDate, Comment)

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

```
SELECT clientno, ViewDate
   FROM  viewing
     WHERE  PropertyNo= 'PG4' AND  comment IS NULL;
```

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees in department 5 whose salary is between $30,000 & $40,000.

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees in department 5 whose salary is between $30,000 & $40,000.

```
SELECT SSN, Fname, Lname
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.DeptNo = D.DNo AND E.salary >= 30000 AND E.salary <= 40000;
```

```
SELECT SSN, Fname, Lname
FROM EMPLOYEE E, DEPARTMENT D
WHERE E.DeptNo = D.DNo AND BETWEEN 30000 AND 40000;
```

ORDER BY clause

Allows the retrieved records to be ordered in ascending (ASC) or descending order (DESC) on any column or combination of columns.

Syntax

```
SELECT {* | [column_expression] [,…]}
   FROM   table_name
     [ORDER BY  column_list [ASC|DESC] ]
```

Single Column ordering

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Produce a list of salaries for all staff, arranged in descending order of salary.

```
SELECT  sno, fname, lname, salary
  FROM  staff
    ORDER BY salary  DESC;
```

Simple Queries
ORDER BY clause

Multiple columns ordering

Property (PropertyNo, Street, City, postcode, Type, OwnerNo, Rooms, Rent)

Produce a list of properties arranged in order of property type and within each property type ordered by rent in descending order.

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

PropertNo	Type	Rooms	Rent
PG16	Flat	4	450
PL94	Flat	4	400
PG36	Flat	3	370
PG4	House	3	650
PA14	House	6	600

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees, ordered by department and, within each department, ordered alphabetically by last name, first name.

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees, ordered by department and, within each department, ordered alphabetically by last name, first name.

```
SELECT DNo, Fname, Lname
FROM EMPLOYEE
WHERE salary >=10000
            ORDER BY DNo DESC
```

```
SELECT DNo, Fname, Lname
FROM EMPLOYEE
    ORDER BY DNo DESC
WHERE salary >=10000
```

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees, ordered by department and, within each department, ordered alphabetically by last name, first name.

```
SELECT DNo, Fname, Lname
FROM EMPLOYEE
WHERE salary >=10000
            ORDER BY DNo DESC
```

```
SELECT DNo, Fname, Lname
FROM (SELECT * FROM EMPLOYEE
    ORDER BY DNo DESC)
WHERE salary >=10000
```

```
SELECT DNo, Fname, Lname
FROM EMPLOYEE
    ORDER BY DNo DESC
WHERE salary >=10000
```

Simple Queries
Aggregation

Functions that operate on a single column of a table and return a single value.

Five aggregation functions defined in SQL:

COUNT	returns the number of rows in a specified column.
SUM	returns the sum of the values in a specified column.
AVG	returns the average of the values in a specified column.
MIN	returns the smallest value in a specified column.
MAX	returns the largest value in a specified column.

Examples:

Property (PropertyNo, Street, City, postcode, Type, OwnerNo, Rooms, Rent)

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

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

count
2

Simple Queries
Aggregation

VIEWING (ClientNo, PropertyNo, ViewDate, Comment)

How many different properties were viewed in May 1998?

```
SELECT  COUNT(DISTINCT  PropertyNo)  AS  count
  FROM  viewing
    WHERE  Viewdate  BETWEEN  '1-May-98'  AND  '31-May-98';
```

count
2

Simple Queries
Aggregation

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Find the total number of Managers and the sum of their salaries.

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

count	sum
2	54000

Simple Queries
Aggregation

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

Find the minimum, maximum, and average staff salary.

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

min	max	avg
9000	30000	17000

Simple Queries
GROUP BY clause

Groups the data from the SELECT table(s) and produces a single summary row for each group.

Example:

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

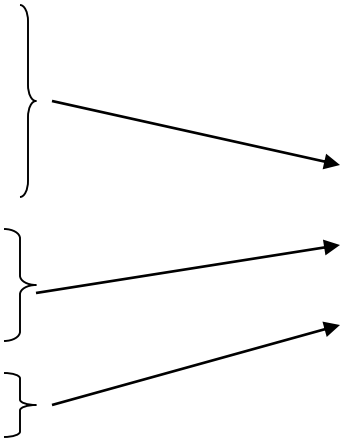
Find the number of staff working in each branch and the sum of their salaries.

```
SELECT  bno, COUNT(sno) AS count, SUM(salary) AS sum
  FROM  staff
    GROUP BY bno;
```

bno	count	sum
B003	3	54000
B005	2	39000
B007	1	9000

Simple Queries
GROUP BY clause

bno	sno	salary
B003	SG37	12000
B003	SG14	18000
B003	SG5	24000
B005	SL21	30000
B005	SL41	9000
B007	SA9	9000



count	sum
3	54000
2	39000
1	9000

Simple Queries
HAVING clause

Designed for use with the GROUP BY clause to restrict the groups that appear in the final result table.
WHERE clause filters individual rows going into the final result table.
HAVING clause filters groups going into the final result table.

Example:

STAFF(sno, fname, lname, position, sex, dob, salary, bno)

For each branch office with more than one member of staff, find the number of staff working in each branch and the sum of their salaries.

```
SELECT  bno, COUNT(sno) AS count, SUM(salary) AS sum
  FROM  staff
    GROUP BY bno
      HAVING COUNT(sno) > 1;
```

bno	count	sum
B003	3	54000
B005	2	39000

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

For each project on which more than two employees work, retrieve the project number and the number of employees who work on the project.

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

For each project on which more than two employees work, retrieve the project number and the number of employees who work on the project.

```
SELECT  PNo, COUNT(SSN) AS count
  FROM  WORKS_ON
    GROUP BY PNo
      HAVING COUNT(SSN) > 1;
```

Additional Relational Operations: Aggregate Functions and Grouping

- A type of request that cannot be expressed in the basic relational algebra is to specify mathematical **aggregate functions** on collections of values from the database.
- Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.
  - These functions are used in simple statistical queries that summarize information from the database tuples.
- Common functions applied to collections of numeric values include
  - SUM, AVERAGE, MAXIMUM, and MINIMUM.
- The COUNT function is used for counting tuples or values.

Aggregate Function Operation

- Use of the Aggregate Functional operation $\mathcal{F}$
  - $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE) computes the count (number) of employees and their average salary
    - Note: count just counts the number of rows, without removing duplicates

Using Grouping with Aggregation

- The previous examples all summarized one or more attributes for a set of tuples
  - Maximum Salary or Count (number of) Ssn
- Grouping can be combined with Aggregate Functions
- Example: For each department, retrieve the DNO, COUNT SSN, and AVERAGE SALARY
- A variation of aggregate operation $\mathcal{F}$ allows this:
  - Grouping attribute placed to left of symbol
  - Aggregate functions to right of symbol
  - $_{\text{DNO}}\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE)
- Above operation groups employees by DNO (department number) and computes the count of employees and average salary per department

Examples of applying aggregate functions and grouping

Figure 6.10

The aggregate function operation.

(a) $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(_{\text{Dno}}\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE}))$.

(b) $_{\text{Dno}}\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

(c) $\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

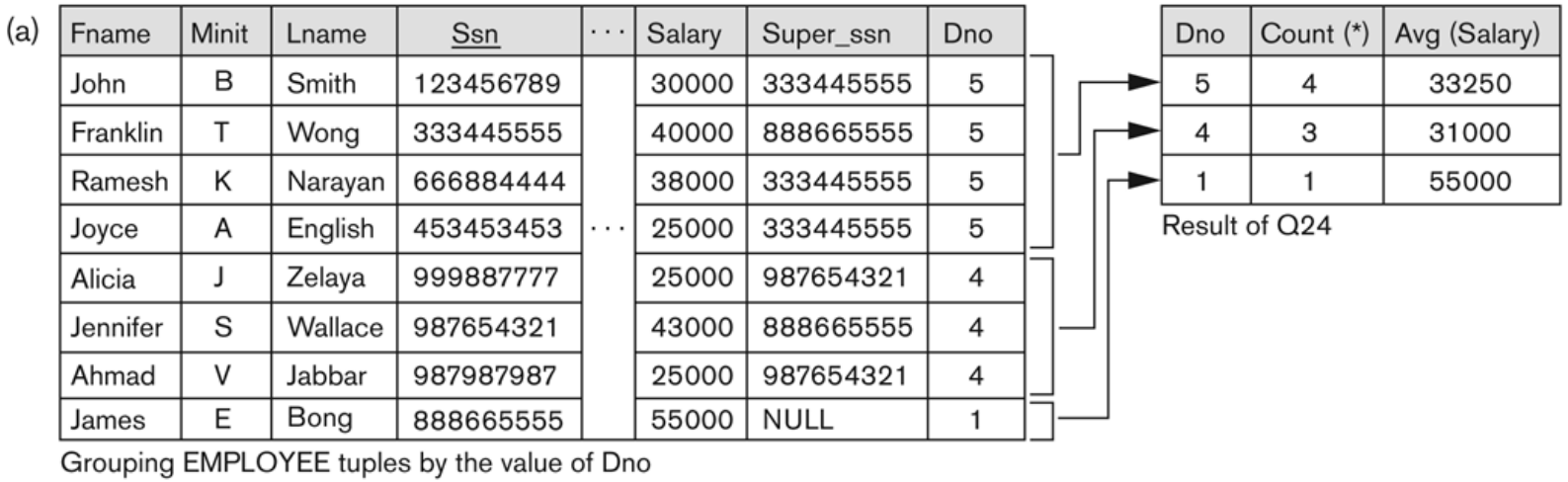
Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

Illustrating aggregate functions and grouping

Figure 8.6
Results of GROUP BY and HAVING. (a) Q24. (b) Q26.



(a)

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

Subqueries

A complete SELECT statement can be embedded (subselect) within another SELECT statement.

A subselect can be used in the WHERE and HAVING clauses of the outer SELECT statement (nested query).

A subquery can be used immediately following a relational operator.

Subquery always enclosed in parentheses.

Type of subquery:

- A *scalar subquery* returns a single column and a single row (singlevalue).
- A *row subquery* returns multiple columns, but a single row.
- A *table subquery* returns one or more columns and multiple rows.

Subqueries

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

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

```
SELECT  sno,  fname,  lname,  position
  FROM staff
    WHERE  bno  = (SELECT  bno
                   FROM  branch
                    WHERE  street = '163 Main St');
```

Subqueries

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)

Example:
List the staff whose salary is greater than the average salary, and list by how much their salary is greater than the average.

```
SELECT sno, fname, lname, position, salary – (SELECT
       avg(salary) FROM staff ) AS sal_diff
  FROM staff
    WHERE  salary  > ( SELECT  avg(salary)
                       FROM  staff  );
```

Subqueries

The following rules apply to subqueries:

- The ORDER BY clause may not be used in a subquery .
- The subquery SELECT list must consist of a single column name or expression, except for subqueries that use the keyword EXISTS.
- By default, column names in a subquery refer to the table name in the FROM clause of the subquery. It is possible to refer to a table in a FROM clause in an outer query by qualifying the column name; in this case the subquery is called a *correlated subquery*.
- When a subquery is one of the two operands involved in a comparison, the subquery must appear on the right-hand side of the comparison.

Subqueries
IN

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Example:
List the properties that are handled by staff who work in the branch at '163 Main St'.

```
SELECT pno, street, area, city, pcode, type, rooms, rent
 FROM property_for_rent
   WHERE sno  IN
       (SELECT  sno
          FROM  staff
            WHERE  bno =
                    (SELECT  bno
                      FROM  branch
                         WHERE  street  =  '163 MainSt'));
```

Question

Assume the following relational schema:

EMPLOYEE(Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT(Dname, DNo)
PROJECT(PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

Show the resulting salaries if every employee working on ‘X’ project is given all %10 raise.

Subqueries
ANY/ ALL

- Used with subqueries that produce a single column of numbers.
- If the subquery is preceded by the keyword ALL, the condition will only be true if it is satisfied by all values produced by the subquery.
- If the subquery is preceded by the keyword ANY or SOME, the condition will be true if it is satisfied by any (one or more) values produced by the subquery.

Subqueries
ANY/ ALL

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Find staff whose salary is larger than the salary of at least one member of staff at branch B3.

```
SELECT  sno,  fname,  lname,  position, salary
  FROM staff
    WHERE salary > SOME
           (SELECT salary
             FROM  staff
              WHERE bno = 'B3');
```

sno	FName	LName	position	salary
SL21	John	White	Manager	30000
SG14	David	Ford	Supervisor	18000
SG5	Susan	Brand	Manager	24000

Subqueries
ANY/ ALL

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Find staff whose salary is larger than the salary of every member of staff at branch B3.

```
SELECT  sno,  fname,  lname,  position, salary
  FROM staff
    WHERE salary > ALL
           (SELECT salary
             FROM  staff
              WHERE bno = 'B3');
```

Sno	FName	LName	position	salary
SL21	John	White	Manager	30000

Question

Assume the following relational schema:

EMPLOYEE (Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT (Dname, DNo)
PROJECT (PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

For each department that has more than 5 employees, retrieve the department number and the number of its employees who are making more than $40,000.

Multi-Table Queries

- So far, the columns that are to appear in the result table must all come from a single table.
- To combine columns from several tables into a result table, we need to use a join operation.
- To perform a join, we include more than one table name in the FROM clause. WHERE clause to specify the join columns.

```
SELECT [DISTINCT|ALL] {* |column |[column_expression
                                   [AS new_name]] [,…]}
   FROM   table_name [alias] [, … ]
     [WHERE  condition];
```

Simple Join

CLIENT (ClientNo, Fname, Lname, telNo, Type, Rent)
VIEWING (ClientNo, PropertyNo, Date, Comment)

Example:
List the 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;
```

Alternatives:

```
FROM  client c  JOIN  viewing  v  ON  c.clientNo = v.clientNo;
FROM  client    JOIN  viewing  USING  clientNo;
FROM  client NATURAL JOIN  viewing;
```

Sorting a Join

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Example:
For each branch office, list the names of staff who manage properties, and the properties they manage, ordered by branch number, staff number and property number.

```
SELECT  s.bno, s.sno, fname,  lname,  pno
  FROM  staff s,  propertyforrent p
    WHERE  s.sno  =  p.sno
     ORDER BY  s.bno, s.sno, p.pno;
```

bno	Sno	FName	LName	pno
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

Question

Assume the following relational schema:

EMPLOYEE (Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT (Dname, DNo)
PROJECT (PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

List all employees and identify the projects they are working on, ordered by department and, within each department, ordered alphabetically by last name, first name.

Three-Table Join

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

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

```
SELECT  b.bno, b.city, s.sno, fname, lname, pno
 FROM  branch b, staff s, propertyForRent p
  WHERE  b.bno = s.bno  AND  s.sno = p.sno;
```

Alternatives:

```
FROM  (Branch b  JOIN  staff s USING  bno)  As  bs
               JOIN  PropertyForRent p  USING  sno;
```

Multiple grouping columns

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Exmaple:
Find the number of properties handled by each staff member and branch.

```
SELECT  s.bno,  s.sno,  COUNT(*)  AS  count
  FROM  staff  s,  propertyForRent  p
    WHERE  s.sno = p.sno
       GROUP BY s.bno, s.sno;
```

bno	Sno	count
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Computing a Join

A join is a subset of the Cartesian product.

The Cartesian product of two tables is another table consisting of all possible pairs of rows from the two table.

The columns of the product table are all the columns of the first table followed by all the columns of the second table.

Format of SELECT statement for the Cartesian product:

```
SELECT [DISTICNT | ALL]  {* | column_list }
  FROM  table_name1 CROSS JOIN  table_name2;
```

Computing a Join

The procedure for generating the results of a SELECT with a join are as follows:

- Form the Cartesian product of the tables named in the FROM clause.
- If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition. In terms of the relational algebra, this operation yields a restriction of the Cartesian product.
- For each remaining row, determine the value of each item in the SELECT list to produce a single row in the result table.
- If SELECT DISTINCT has been specified, eliminate any duplicate rows from the result table.
- If there is an ORDER BY clause, sort the result table as required.

Outer Join

The **join** operation combines data from two tables by forming pairs of related rows where the matching columns in each table have the same value. If one row of a table is unmatched, the row is omitted from the result table.

Outer join include the unmatched rows in the result table.

Three types of outer join:

- Left
- Right
- Full

Join Example

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
B003	Glasgow	PG4	Glasgow
B002	London	PL94	London

```
SELECT   b.*, p.*
  FROM  branch  b, property p
    WHERE  b.bcity =  p.pcity;
```

Left Outer Join

Example:
List the branch offices and properties that are in the same city along with any unmatched branches.

```
SELECT  b.*, p.*
 FROM  branch  b
    LEFT  JOIN  property  p ON
          b.bcity  =  p.pcity;
```

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT  b.*, p.*
 FROM  branch  b
  LEFT  JOIN  property  p ON
            b.bcity  =  p.pcity;
```

Right Outer Join

Example:
List the branch offices and properties in the same city and any unmatched property.

```
SELECT  b.*, p.*
  FROM  branch  b
     RIGHT  JOIN  property  p ON
              b.bcity  =  p.pcity;
```

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PL94	London
B002	London	PG4	Glasgow

```
SELECT  b.*, p.*
  FROM  branch  b
     RIGHT  JOIN  property  p ON
              b.bcity  =  p.pcity;
```

Full Outer Join

Example:
List the branch offices and properties that are in the same city and any unmatched branches or properties.

```
SELECT  b.*, p.*
  FROM  branch  b
     FULL  JOIN  property  p ON
              b.bcity  =  p.pcity;
```

BRANCH

BranchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

PROPERTY

PropertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

BranchNo	bCity	PropertyNo	pCity
NULL	NULL	PA14	Aberdeen
B003	Glasgow	PG4	Glasgow
B004	Bristol	NULL	NULL
B002	London	PL94	London

```
SELECT  b.*, p.*
  FROM  branch  b
     FULL  JOIN  property  p ON
              b.bcity  =  p.pcity;
```

EXIST/ NOT EXIST

Used only with correlated subqueries.
EXISTS is true if and only if there exists at least one row in the result table returned by the subquery. It is false if the subquery returns an empty result table.

Example:

STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Find all staff who work in a London branch.

```
SELECT  sno,  fname,  lname,  position
  FROM  staff  s
   WHERE  EXISTS
     (SELECT  *
       FROM  branch  b
         WHERE  s.bno  = b.bno  AND  city  = 'London');
```

Question

Assume the following relational schema:

EMPLOYEE (Fname, Lname, SSN, DOB, Address, Sex, salary, DeptNo)
DEPARTMENT (Dname, DNo)
PROJECT (PName, PNo, PLocation, Dno)
WORKS_ON(SSN, PNo, Hours)

Retrieve the names of employees who works on no project.

UNION

PROPERTYFORRENT (pno, street, area, city, pcode, type, rooms, rent, sno)
STAFF (sno, fname, lname, position, sex, DOB, salary, bno)
BRANCH (bno, street, city, postcode)

Example:
Construct a list of all cities where there is either a branch office or a rental property.

```
(SELECT  city   FROM  branch)
UNION
(SELECT  city   FROM  propertyforrent);
```

INTERSECT

Example:
Construct a list of all cities where there is both a branch office and a rental property.

```
(SELECT  city   FROM  branch)
INTERSECT
(SELECT  city   FROM propertyforrent);


SELECT DISTINCT b.city
  FROM  branch  b, propertyforrent  p
    WHERE  b.city=p.city;


SELECT DISTINCT city
  FROM   branch  b
   WHERE  EXISTS
            (SELECT  *
                  FROM propertyforrent  p
                     WHERE  p.city  =  b.city);
```

EXCEPT

Example:
Construct a list of all cities where there is a branch office but no rental property.

```
(SELECT  city  FROM  branch)
EXCEPT
(SELECT  city  FROM propertyforrent);


SELECT  DISTINCT city
  FROM  branch
   WHERE  city NOT IN
          (SELECT   city
            FROM propertyforrent);


SELECT DISTINCT city
  FROM   branch  b
   WHERE  NOT EXISTS
           (SELECT  *  FROM propertyforrent  p
                    WHERE  p.city  =  b.city);
```

Adding Data to DB (INSERT)

Syntax

```
INSERT INTO table_name [(column (,…))]
     { VALUES (date_value (,…)) |  subquery };
```

- *table_name* may be either a base table or an updatable view.
- *column_list* represents a list of one or more column names separated by commas.
- If omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- If specified, then any columns that are omitted from the list must have been declared as NULL column.
- *data_value* must match the *column_list* as follows:
  - The number of items in each list must be same.
  - There must be a direct correspondence in the position of items in the two lists, so that the first item in the *data_value_list* applies to the first item in the *column_list*, and so on.
  - The data type of each item in the *data_value_list* must be compatible with the data type of the corresponding column.

Simple INSERT

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Insert a new row into the staff table supplying data for all columns.

```
INSERT INTO staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 'M',
        DATE '1957-05-25', 8300, 'B003');
```

Simple INSERT

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Insert a new row into the staff table supplying data for all mandatory columns, knowing that the sex and birth date are optional fields.

```
INSERT INTO staff (Sno, fname, lname, position, salary, bno)
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', 8300, 'B003');
```

Alternative:

```
INSERT INTO staff
VALUES ('SG16', 'Alan', 'Brown', 'Assistant', NULL, NULL, 8300,
        'B003');
```

INSERT with subqueries

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)
PROPERTYFORRENT(Pno, street, city, postcode, type, rooms, rent, ono, sno, bno)
StaffPropCount(sno, fname, lname, propcount)

Example:
Insert rows into the StaffPropCount table using the staff and property_for_rent tables.

```
INSERT INTO staffPropCount
(SELECT  s.sno, fname, lname, COUNT(*)
  FROM  staff s, PropertyForRent p
   WHERE  s.sno = p.sno
    GROUP BY s.sno, fname, lname)
UNION
(SELECT  sno, fname, lname, 0
  FROM  Staff
    WHERE  sno NOT IN (SELECT DISTINCT sno
                       FROM  PropertyForRent));
```

Sno	FName	LName	propCount
SG14	David	Ford	1
SL21	John	White	0
SG37	Ann	Beech	2
SA9	Mary	Howe	1
SG5	Susan	Brand	0
SL41	Julie	Lee	1

Modifying Data in the DB (UPDATE)

Syntax

```
    UPDATE table_name
       SET column_name1 = data_value1 [, column_namei =
data_valuei ...]
            [WHERE  search_condition]
```

- *table_name* may be either a base table or an updatable view.
- The SET clause specifies the names of one or more columns that are updated for all rows in the table.
- Only rows that satisfy the *search_condition* are updated.
- *data_values* must be compatible with the data types for the corresponding columns.

Simple UPDATE

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Give all staff a 3% pay increase.

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

Example:
Give all managers a 3% pay increase.

```
UPDATE  staff
  SET  salary = salary * 1.03
    WHERE  position = 'Manager';
```

Simple UPDATE

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Promote David Ford (sno = ‘SG14’) to Manager and change his salary to $18,000.

```
UPDATE  staff
  SET  position=‘Manager’, salary = 18000
   WHERE  sno=‘SG14’;
```

Deleting Data from the DB (DELETE)

Syntax

```
DELETE  FROM  table_name
        [WHERE  search_condition];
```

- *table_name* may be either a base table or an updatable view.
- Only rows that satisfy the *search_condition* are deleted.
- If no *search_condition* is omitted, all rows are deleted from the table.
- DELETE does not delete the table itself, only rows in the table.

Simple DELETE

STAFF(sno, fname, lname, position, sex, DOB, salary, bno)

Example:
Delete all staff in branch B003.

```
DELETE FROM  staff
  WHERE  bno = 'B003';
```

Example:
Delete all staff.

```
DELETE FROM staff;
```