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.
- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.
- Until SQL:1999, 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.
- SQL is relatively easy to learn:
  - it is non-procedural - you specify what information you require, rather than how to get it;
  - it is essentially free-format.

Objectives of SQL

- Consists of standard English words:

```
1) CREATE TABLE Staff(staffNo VARCHAR(5),
        lName VARCHAR(15),
                        salary DECIMAL(7,2));
2) INSERT INTO Staff VALUES ('SG16', 'Brown', 8300);
3) 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.

History of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called ‘Structured English Query Language’ (SEQUEL).
- A revised version, SEQUEL/2, was defined in 1976 but name was subsequently changed to SQL for legal reasons.
- Still pronounced ‘see-quel’, though official pronunciation is ‘S-Q-L’.
- IBM subsequently produced a prototype DBMS called System R, based on SEQUEL/2.
- Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.

History of SQL

- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.
- In 1987, ANSI and ISO published an initial standard for SQL.
- In 1989, ISO published an addendum that defined an ‘Integrity Enhancement Feature’.
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.
- In 1999, SQL:1999 was released with support for object-oriented data management.
- In late 2003, SQL:2003 was released.
- In summer 2008, SQL:2008 was released.
- In late 2011, SQL:2011 was released.

Importance of SQL

- SQL has become part of application architectures such as IBM's Systems Application Architecture.
- It is strategic choice of many large and influential organizations (e.g. X/OPEN).
- SQL is Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to American Government.
- SQL is used in other standards and even influences development of other standards as a definitional tool. Examples include:
  - ISO's Information Resource Directory System (IRDS) Standard
  - Remote Data Access (RDA) Standard.

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.
- 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

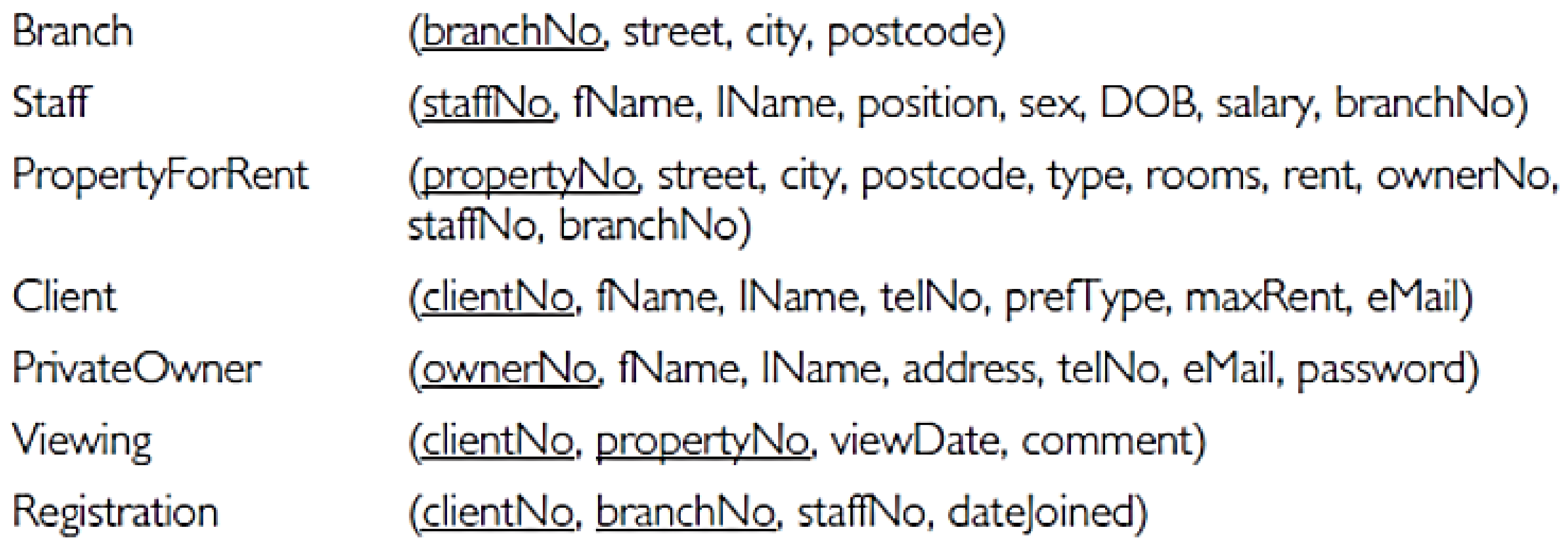
```
SELECT [DISTINCT | ALL]
    {* | [columnExpression [AS newName]] [,...] }
FROM                TableName [alias] [, ...]
[WHERE     condition]
[GROUP BY  columnList]  [HAVING    condition]
[ORDER BY  columnList]
```

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.

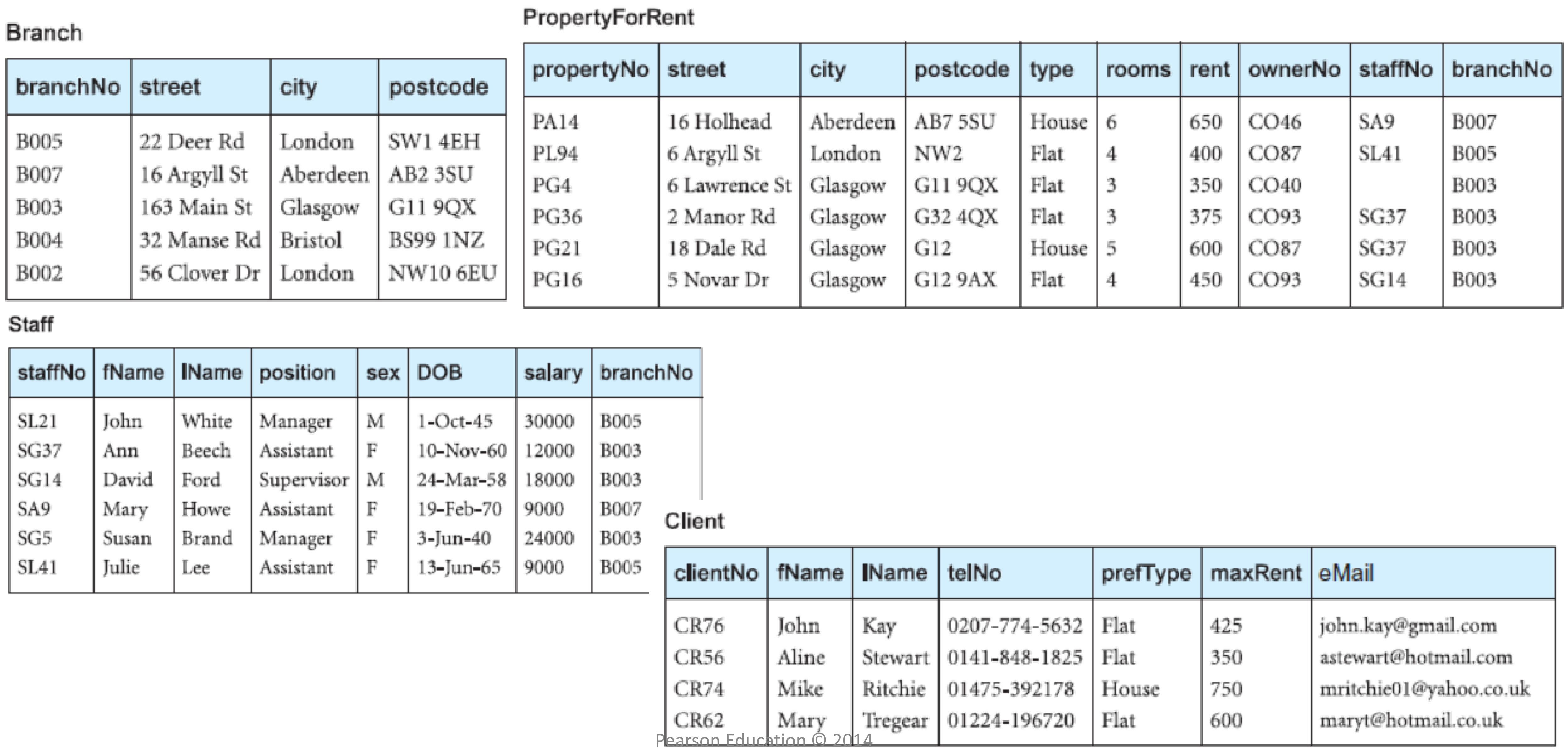
Relational Database Schemas

- The relational schema for part of the *DreamHome* case study is:



Branch	(branchNo, street, city, postcode)
Staff	(staffNo, fName, lName, position, sex, DOB, salary, branchNo)
PropertyForRent	(propertyNo, street, city, postcode, type, rooms, rent, ownerNo, staffNo, branchNo)
Client	(clientNo, fName, lName, telNo, prefType, maxRent, eMail)
PrivateOwner	(ownerNo, fName, lName, address, telNo, eMail, password)
Viewing	(clientNo, propertyNo, viewDate, comment)
Registration	(clientNo, branchNo, staffNo, dateJoined)

Instance of relational schema



Branch

branchNo	street	city	postcode
B005	22 Deer Rd	London	SW1 4EH
B007	16 Argyll St	Aberdeen	AB2 3SU
B003	163 Main St	Glasgow	G11 9QX
B004	32 Manse Rd	Bristol	BS99 1NZ
B002	56 Clover Dr	London	NW10 6EU

PropertyForRent

propertyNo	street	city	postcode	type	rooms	rent	ownerNo	staffNo	branchNo
PA14	16 Holhead	Aberdeen	AB7 5SU	House	6	650	CO46	SA9	B007
PL94	6 Argyll St	London	NW2	Flat	4	400	CO87	SL41	B005
PG4	6 Lawrence St	Glasgow	G11 9QX	Flat	3	350	CO40		B003
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375	CO93	SG37	B003
PG21	18 Dale Rd	Glasgow	G12	House	5	600	CO87	SG37	B003
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450	CO93	SG14	B003

Staff

staffNo	fName	lName	position	sex	DOB	salary	branchNo
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	3-Jun-40	24000	B003
SL41	Julie	Lee	Assistant	F	13-Jun-65	9000	B005

Client

clientNo	fName	lName	telNo	prefType	maxRent	eMail
CR76	John	Kay	0207-774-5632	Flat	425	john.kay@gmail.com
CR56	Aline	Stewart	0141-848-1825	Flat	350	astewart@hotmail.com
CR74	Mike	Ritchie	01475-392178	House	750	mritchie01@yahoo.co.uk
CR62	Mary	Tregear	01224-196720	Flat	600	maryt@hotmail.co.uk

Instance of relational schema

PrivateOwner

ownerNo	fName	lName	address	telNo	eMail	password
CO46	Joe	Keogh	2 Fergus Dr, Aberdeen AB2 7SX	01224-861212	jkeogh@lhh.com	********
CO87	Carol	Farrel	6 Achray St, Glasgow G32 9DX	0141-357-7419	cfarrel@gmail.com	********
CO40	Tina	Murphy	63 Well St, Glasgow G42	0141-943-1728	tinam@hotmail.com	********
CO93	Tony	Shaw	12 Park Pl, Glasgow G4 0QR	0141-225-7025	tony.shaw@ark.com	********

Viewing

clientNo	propertyNo	viewDate	comment
CR56	PA14	24-May-13	too small
CR76	PG4	20-Apr-13	too remote
CR56	PG4	26-May-13	
CR62	PA14	14-May-13	no dining room
CR56	PG36	28-Apr-13	

Registration

clientNo	branchNo	staffNo	dateJoined
CR76	B005	SL41	2-Jan-13
CR56	B003	SG37	11-Apr-12
CR74	B003	SG37	16-Nov-11
CR62	B007	SA9	7-Mar-12

SELECT Statement

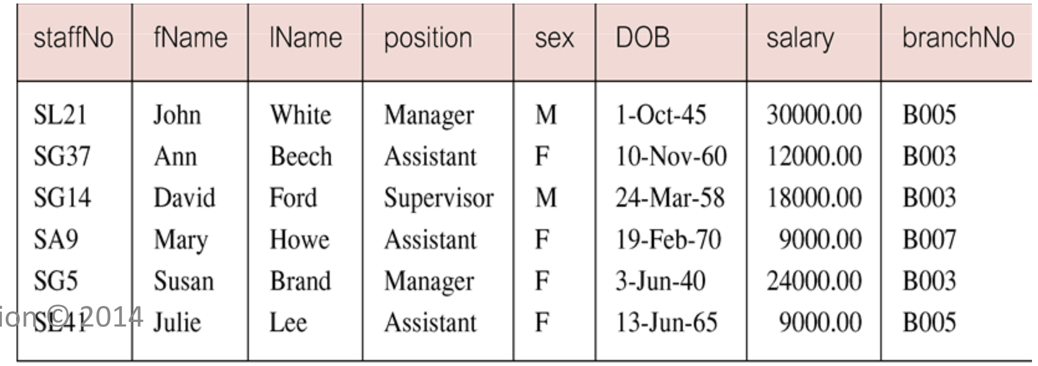
- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.
- Example: All Columns, All Rows

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;
```



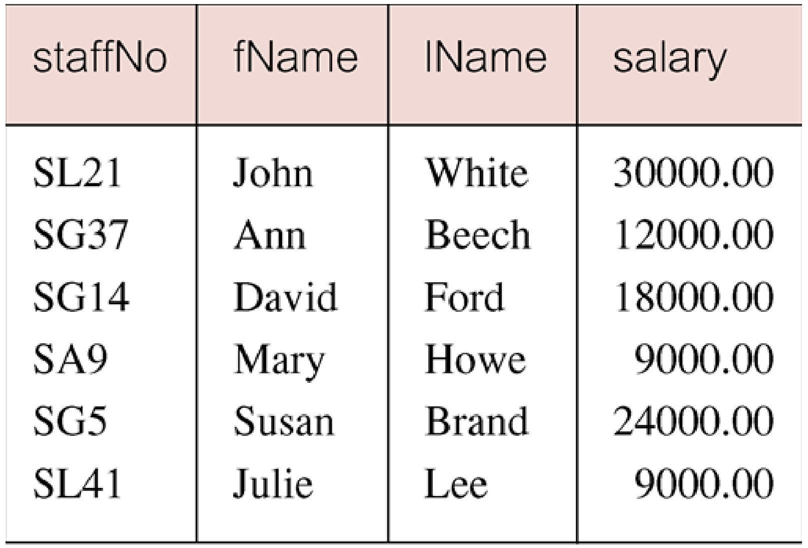
staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000.00	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SA9	Mary	Howe	Assistant	F	19-Feb-70	9000.00	B007
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003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

SELECT Statement: Specific Columns, All Rows

Example: 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;
```



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

SELECT Statement: DISTINCT

- Example 6.3 Use of DISTINCT

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

```
SELECT propertyNo
FROM Viewing;
```

propertyNo
PA14
PG4
PG4
PA14
PG36

Use DISTINCT to eliminate duplicates

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

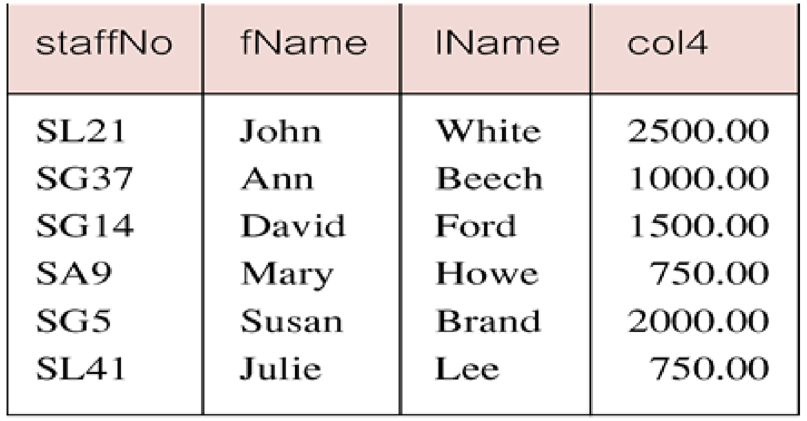
propertyNo
PA14
PG4
PG36

SELECT Statement: Calculated Fields

- Example: Calculated Fields

  Produce list of monthly salaries for all staff, showing staff number, first/last name, and salary.

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



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

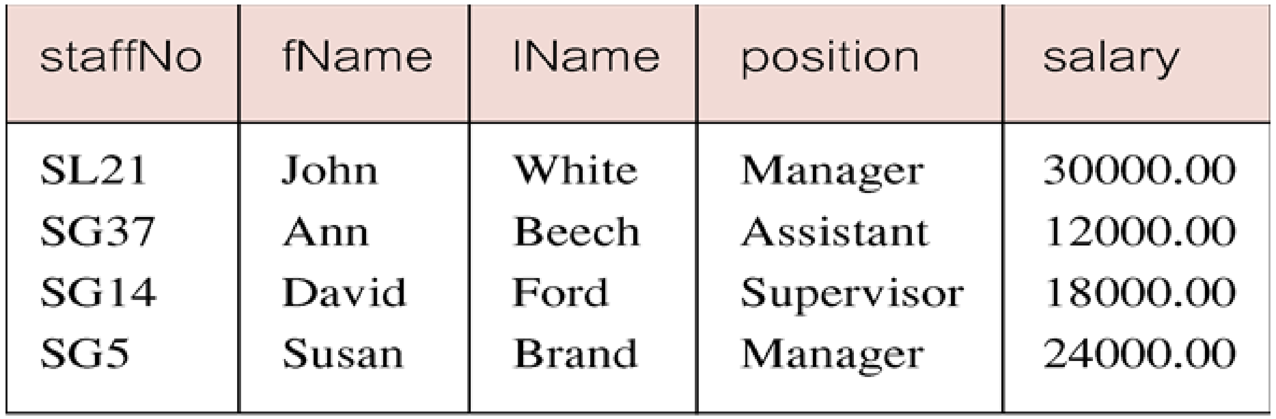
- To name column, use AS clause:

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

SELECT Statement: Comparison Search Condition

- Example: Comparison Search Condition
  List all staff with a salary greater than 10,000.

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



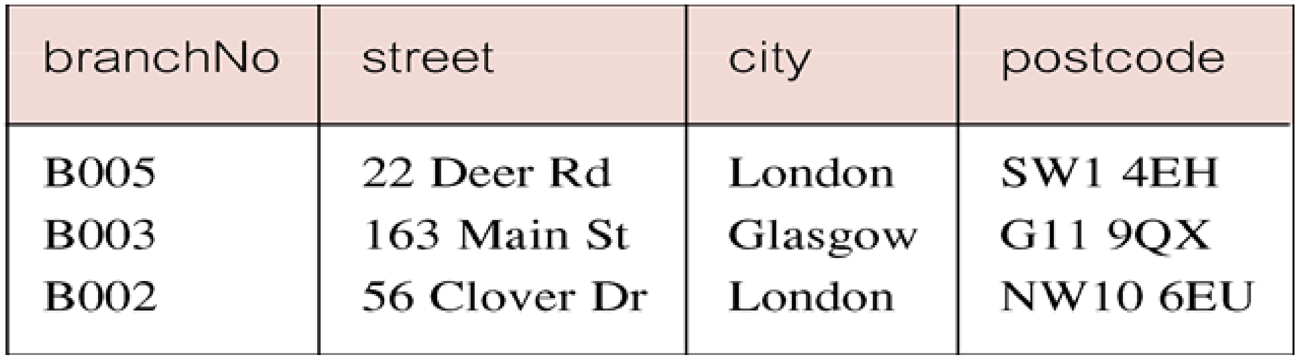
staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

SELECT Statement: Compound Comparison Search Condition

- Example: Compound Comparison Search Condition

  List addresses of all branch offices in London or Glasgow.

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



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

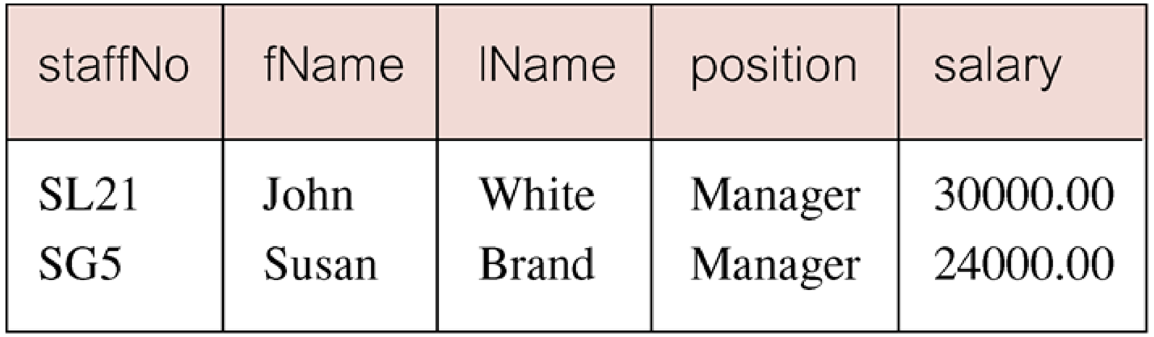
SELECT Statement: Range Search Condition

- Example: 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.



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

SELECT Statement: Range Search Condition

- Example: 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;
```

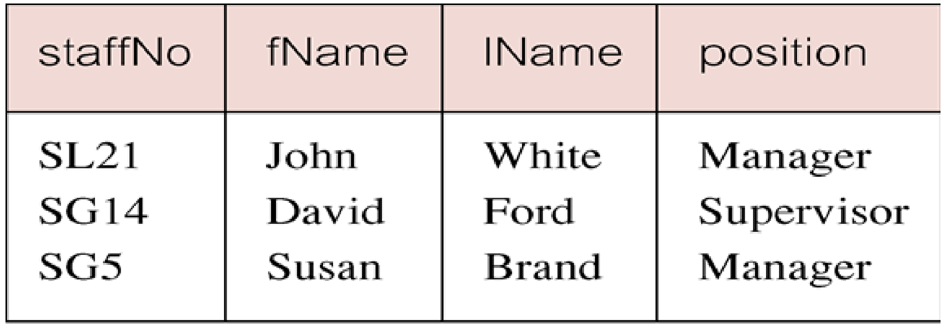
- Useful, though, for a range of values.

SELECT Statement: Set Membership

- Example: Set Membership

  List all managers and supervisors.

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



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

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

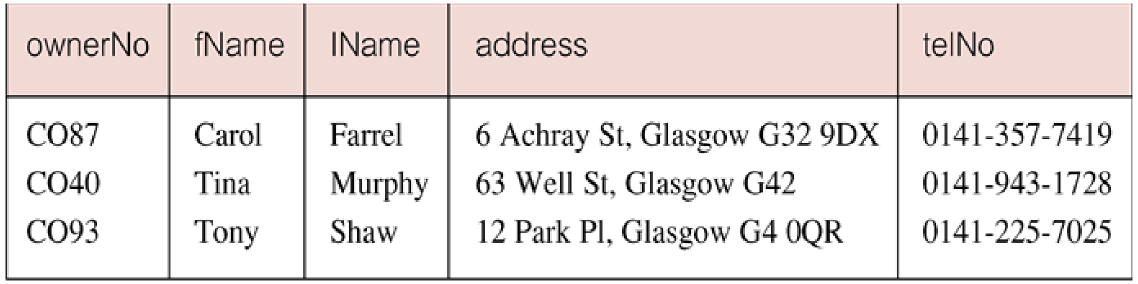
```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position = ‘Manager’ OR position = ‘Supervisor’;
```

- IN is more efficient when set contains many values.

SELECT Statement: Pattern Matching

- Example: Pattern Matching
  Find all owners with the string ‘Glasgow’ in their address.

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



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

- 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’.

SELECT Statement: NULL Search Condition

- Example: 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;
```

clientNo	viewDate
CR56	26-May-04

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

SELECT Statement: Single Column Ordering

- Example: 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;
```

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

SELECT Statement: Example 6.12 Multiple Column Ordering

- Example: Multiple Column Ordering
  Produce abbreviated list of properties in order of property type.

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

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

- 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;
```

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

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 returns a single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.
- 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 out with an aggregate function. For example, the following is illegal:

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

Use of COUNT(*)

- Example:

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

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

myCount
5

- How many different properties viewed in May '13?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN '1-May-13'
          AND '31-May-13';
```

myCount
2

Use of COUNT and SUM

- Example:

  Find number of Managers and sum of their salaries.

```
SELECT COUNT(staffNo) AS myCount, SUM(salary) AS mySum
FROM Staff
WHERE position = ‘Manager’;
```

myCount	mySum
2	54000.00

Use of MIN, MAX, AVG

- Example:

Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS myMin,
MAX(salary) AS myMax,
AVG(salary) AS myAvg
FROM Staff;
```

myMin	myMax	myAvg
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.

- 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.

Use of GROUP BY

- Example:

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

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

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
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.

Use of HAVING

- Example:
- 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 myCount,  SUM(salary) AS mySum
FROM Staff
GROUP BY branchNo
HAVING COUNT(staffNo) > 1
ORDER BY branchNo;
```

branchNo	myCount	mySum
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 DELETE statements.

Subquery with Equality

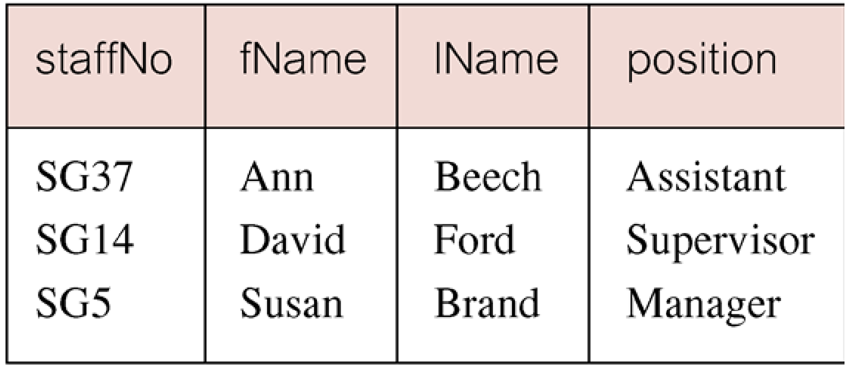
- Example:

  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’);
```

- 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’;
```



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

Subquery with Aggregate

- Example:

  List all staff whose salary is greater than the average salary, and show by how much.

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

- 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 – 17000 As salDiff
FROM Staff
WHERE salary > 17000;
```

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.
- 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.

Nested subquery: use of IN

- Example:
  List properties handled by staff at ‘163 Main St’.

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
FROM PropertyForRent
WHERE staffNo IN
      (SELECT staffNo
       FROM Staff
       WHERE branchNo =
                         (SELECT branchNo
                          FROM Branch
                          WHERE street = ‘163 Main St’));
```

propertyNo	street	city	postcode	type	rooms	rent
PG16	5 Novar Dr	Glasgow	G12 9AX	Flat	4	450
PG36	2 Manor Rd	Glasgow	G32 4QX	Flat	3	375
PG21	18 Dale Rd	Glasgow	G12	House	5	600

ANY and ALL

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- With ALL, condition will only be true if it is satisfied by all values produced by subquery.
- With ANY, condition will be true if it is satisfied by any values produced by subquery.
- If subquery is empty, ALL returns true, ANY returns false.
- SOME may be used in place of ANY.

Use of ANY/SOME

- Example:

  Find staff whose salary is larger than salary of at least one member of staff at branch B003.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > SOME (SELECT salary
                     FROM Staff
                     WHERE branchNo = 'B003');
```

- Inner query produces set {12000, 18000, 24000} and outer query selects those staff whose salaries are greater than any of the values in this set.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Use of ALL

- Example:

  Find staff whose salary is larger than salary of every member of staff at branch B003.

```
SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary > ALL(SELECT salary
                   FROM Staff
                   WHERE branchNo = 'B003');
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

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).
- 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: 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;
```

- 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.

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: 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;
```

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: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, PropertyForRent p
WHERE b.branchNo = s.branchNo AND s.staffNo = p.staffNo
ORDER BY b.branchNo, s.staffNo, propertyNo;
```

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: Multiple Grouping Columns

- Find number of properties handled by each staff member.

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

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

Computing a Join

- Procedure for generating results of a join are:
  - Form Cartesian product of the tables named in 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.
  - For each remaining row, determine value of each item in SELECT list to produce a single row in result table.4
  - If DISTINCT has been specified, eliminate any duplicate rows from the result table.
  - 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

- The (inner) join of these two tables:

```
SELECT b.*, p.*
FROM Branch1 b, PropertyForRent1 p
WHERE b.bCity = p.pCity;
```

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

Outer Joins

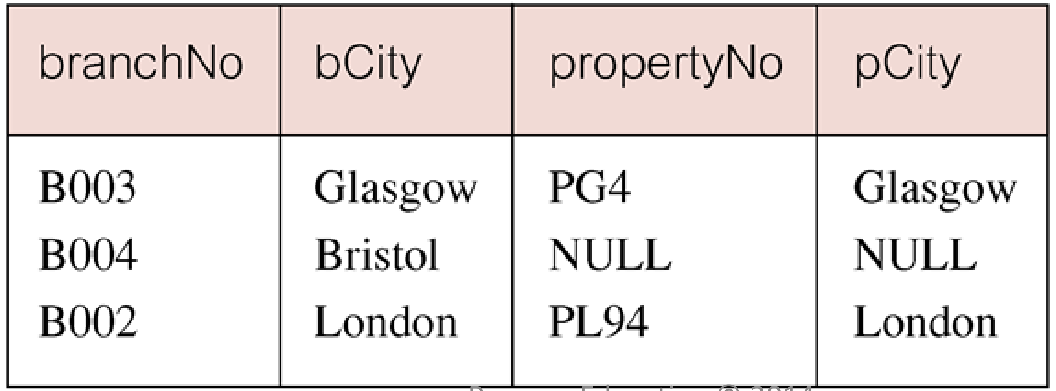
- Result table has two rows where cities are same.
- There are no rows corresponding to branches in Bristol and Aberdeen.
- To include unmatched rows in result table, use an Outer join.

Example: Left Outer Join

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

```
SELECT b.*, p.*
FROM Branch1 b LEFT JOIN
        PropertyForRent1 p ON b.bCity = p.pCity;
```

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



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

Example: Right Outer Join

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

```
SELECT b.*, p.*
FROM Branch1 b RIGHT JOIN PropertyForRent1 p ON b.bCity = p.pCity;
```

- 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.

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

Example: Full Outer Join

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

```
SELECT b.*, p.*
FROM Branch1 b FULL JOIN  PropertyForRent1 p ON b.bCity = p.pCity;
```

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

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.
- 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: 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');
```

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

Example: 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 always be true and query would be:

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

- 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';
```

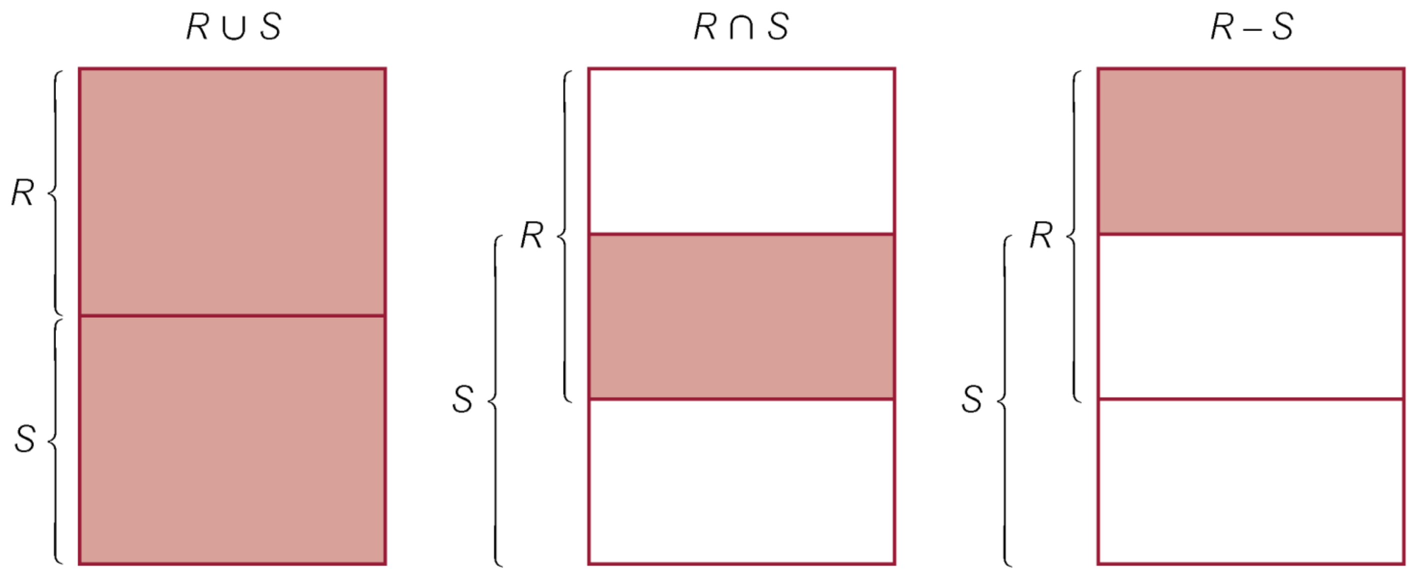
Union, Intersect, and Difference (Except)

- Can use normal set operations of Union, Intersection, and Difference to combine results of two or more queries into a single result table.
- Union of two tables, A and B, is table containing all rows in either A or B or both.
- Intersection is table containing all rows common to both A and B.
- Difference is table containing all rows in A but not in B.
- Two tables must be union compatible.
- Format of set operator clause in each case is:

```
op [ALL] [CORRESPONDING [BY {column1 [, ...]}]]
```

- If CORRESPONDING BY specified, set operation performed on the named column(s).
- If CORRESPONDING specified but not BY clause, operation performed on common columns.
- If ALL specified, result can include duplicate rows.

Union, Intersect, and Difference (Except)



(a) Union (b) Intersection (c) Difference

Example: Use of UNION

- List all cities where there is either a branch office or a property.

```
(SELECT city
FROM Branch
WHERE city IS NOT NULL) UNION
(SELECT city
FROM PropertyForRent
WHERE city IS NOT NULL);
```

Or

```
(SELECT *
FROM Branch
WHERE city IS NOT NULL)
UNION CORRESPONDING BY city
(SELECT *
FROM PropertyForRent
WHERE city IS NOT NULL);
```

city
London
Glasgow
Aberdeen
Bristol

- Produces result tables from both queries and merges both tables together.

Example: Use of INTERSECT

- List all cities where there is both a branch office and a property.

```
(SELECT city FROM Branch)
INTERSECT
(SELECT city FROM PropertyForRent);
```

Or

```
(SELECT * FROM Branch)
INTERSECT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```

city
Aberdeen
Glasgow
London

Example: Use of INTERSECT

- Could rewrite this query without INTERSECT operator:

```
SELECT b.city
FROM Branch b PropertyForRent p
WHERE b.city = p.city;
```

Or:

```
SELECT DISTINCT city FROM Branch b
WHERE EXISTS
          (SELECT * FROM PropertyForRent p
          WHERE p.city = b.city);
```

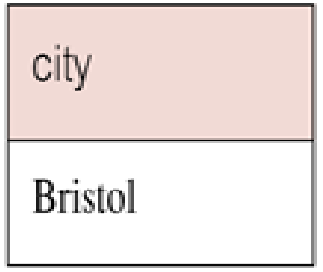
Example: Use of EXCEPT

- List of all cities where there is a branch office but no properties.

```
(SELECT city FROM Branch)
EXCEPT
(SELECT city FROM PropertyForRent);

Or

(SELECT * FROM Branch)
EXCEPT CORRESPONDING BY city
(SELECT * FROM PropertyForRent);
```



city
Bristol

Example: Use of EXCEPT

- Could rewrite this query without EXCEPT:

```
SELECT DISTINCT city FROM Branch
WHERE city NOT IN
        (SELECT city FROM PropertyForRent);
```

Or

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

INSERT

- INSERT INTO TableName [(columnList)]
  VALUES (dataValueList)

- columnList is optional; if omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.
- 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: INSERT … VALUES

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

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

Example: 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’);
```

Or

```
INSERT INTO Staff
VALUES (‘SG44’, ‘Anne’, ‘Jones’, ‘Assistant’, NULL,
             NULL, 8100, ‘B003’);
```

INSERT … SELECT

- Second form of INSERT allows multiple rows to be copied from one or more tables to another:

```
INSERT INTO TableName [ (columnList) ]
SELECT ...
```

Example: INSERT … SELECT

- Assume there is a table StaffPropCount that contains names of staff and number of properties they manage:

  StaffPropCount(staffNo, fName, lName, propCnt)

- Populate StaffPropCount using Staff and PropertyForRent tables.
- Example:

```
INSERT INTO StaffPropCount
     (SELECT s.staffNo, fName, lName, COUNT(*)
     FROM Staff s, PropertyForRent p
     WHERE s.staffNo = p.staffNo
     GROUP BY s.staffNo, fName, lName)
     UNION
     (SELECT staffNo, fName, lName, 0
     FROM Staff
     WHERE staffNo NOT IN
               (SELECT DISTINCT staffNo
               FROM PropertyForRent));
```

staffNo	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

- If second part of UNION is omitted, excludes those staff who currently do not manage any properties.

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.
- 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: 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: 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: 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;
```

SQL – Data Definition

ISO SQL Data Types

DATA TYPE	DECLARATIONS				
boolean	BOOLEAN				
character	CHAR	VARCHAR			
bit[†]	BIT	BIT VARYING			
exact numeric	NUMERIC	DECIMAL	INTEGER	SMALLINT	BIGINT
approximate numeric	FLOAT	REAL	DOUBLE PRECISION		
datetime	DATE	TIME	TIMESTAMP		
interval	INTERVAL				
large objects	CHARACTER LARGE OBJECT		BINARY LARGE OBJECT		

[†]BIT and BIT VARYING have been removed from the SQL:2003 standard.

Integrity Enhancement Feature

- Consider five types of integrity constraints:
  - required data
  - domain constraints
  - entity integrity
  - referential integrity
  - general constraints.

Integrity Enhancement Feature

- Required Data

```
position  VARCHAR(10)     NOT NULL
```

- Domain Constraints

(a) CHECK

```
sex        CHAR     NOT NULL      CHECK (sex IN ('M', 'F'))
```

(b) CREATE DOMAIN

- CREATE DOMAIN DomainName [AS] dataType
- [DEFAULT defaultOption]
- [CHECK (searchCondition)]

For example:

```
CREATE DOMAIN SexType AS CHAR
        CHECK (VALUE IN ('M', 'F'));
        sex       SexType          NOT NULL
```

Integrity Enhancement Feature (IEF)

- searchCondition can involve a table lookup:

```
CREATE DOMAIN BranchNo AS CHAR(4)
CHECK (VALUE IN (SELECT branchNo
                         FROM Branch));
```

- Domains can be removed using DROP DOMAIN:

```
DROP DOMAIN DomainName
        [RESTRICT | CASCADE]
```

IEF - Entity Integrity

- Primary key of a table must contain a unique, non-null value for each row.
- ISO standard supports FOREIGN KEY clause in CREATE and ALTER TABLE statements:

```
PRIMARY KEY(staffNo)
PRIMARY KEY(clientNo, propertyNo)
```

- Can only have one PRIMARY KEY clause per table. Can still ensure uniqueness for alternate keys using UNIQUE:

```
UNIQUE(telNo)
```

IEF - Referential Integrity

- FK is column or set of columns that links each row in child table containing foreign FK to row of parent table containing matching PK.
- Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table.
- ISO standard supports definition of FKs with FOREIGN KEY clause in CREATE and ALTER TABLE:

```
FOREIGN KEY(branchNo) REFERENCES Branch
```

- Any INSERT/UPDATE attempting to create FK value in child table without matching CK value in parent is rejected.
- Action taken attempting to update/delete a CK value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:

CASCADE	- SET NULL
SET DEFAULT	- NO ACTION

IEF - Referential Integrity

- CASCADE: Delete row from parent and delete matching rows in child, and so on in cascading manner.
- SET NULL: Delete row from parent and set FK column(s) in child to NULL. Only valid if FK columns are NOT NULL.
- SET DEFAULT: Delete row from parent and set each component of FK in child to specified default. Only valid if DEFAULT specified for FK columns.
- NO ACTION: Reject delete from parent. Default.

```
FOREIGN KEY (staffNo) REFERENCES Staff        ON DELETE SET NULL

FOREIGN KEY (ownerNo) REFERENCES Owner        ON UPDATE CASCADE
```

IEF - General Constraints

- Could use CHECK/UNIQUE in CREATE and ALTER TABLE.
- Similar to the CHECK clause, also have:

```
CREATE ASSERTION AssertionName
        CHECK (searchCondition)

CREATE ASSERTION StaffNotHandlingTooMuch
   CHECK (NOT EXISTS   (SELECT staffNo
                               FROM PropertyForRent
                               GROUP BY staffNo
                               HAVING COUNT(*) > 100))
```

Data Definition

- SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.
- Main SQL DDL statements are:

CREATE SCHEMA	DROP SCHEMA
CREATE/ALTER DOMAIN	DROP DOMAIN
CREATE/ALTER TABLE	DROP TABLE
CREATE VIEW	DROP VIEW

- Many DBMSs also provide:

CREATE INDEX	DROP INDEX

Data Definition

- Relations and other database objects exist in an environment.
- Each environment contains one or more catalogs, and each catalog consists of set of schemas.
- Schema is named collection of related database objects.
- Objects in a schema can be tables, views, domains, assertions, collations, translations, and character sets. All have same owner.

CREATE SCHEMA

```
CREATE SCHEMA [Name |
               AUTHORIZATION CreatorId ]
DROP SCHEMA Name [RESTRICT | CASCADE ]
```

- With RESTRICT (default), schema must be empty or operation fails.
- With CASCADE, operation cascades to drop all objects associated with schema in order defined above. If any of these operations fail, DROP SCHEMA fails.

CREATE TABLE

```
CREATE TABLE TableName
{(colName dataType [NOT NULL] [UNIQUE]
[DEFAULT defaultOption]
[CHECK searchCondition] [,...]}
[PRIMARY KEY (listOfColumns),]
{[UNIQUE (listOfColumns),] [...,]}
{[FOREIGN KEY (listOfFKColumns)
  REFERENCES ParentTableName [(listOfCKColumns)],
  [ON UPDATE referentialAction]
  [ON DELETE referentialAction ]] [,...]}
 {[CHECK (searchCondition)] [,...] })
```

CREATE TABLE

- Creates a table with one or more columns of the specified dataType.
- With NOT NULL, system rejects any attempt to insert a null in the column.
- Can specify a DEFAULT value for the column.
- Primary keys should always be specified as NOT NULL.
- FOREIGN KEY clause specifies FK along with the referential action.

Example: CREATE TABLE

```
CREATE DOMAIN OwnerNumber AS VARCHAR(5)
   CHECK (VALUE IN (SELECT ownerNo FROM PrivateOwner));
CREATE DOMAIN StaffNumber AS VARCHAR(5)
   CHECK (VALUE IN (SELECT staffNo FROM Staff));
CREATE DOMAIN PNumber AS VARCHAR(5);
CREATE DOMAIN PRooms AS SMALLINT;
     CHECK(VALUE BETWEEN 1 AND 15);
CREATE DOMAIN PRent AS DECIMAL(6,2)
     CHECK(VALUE BETWEEN 0 AND 9999.99);
```

Example: CREATE TABLE

```
CREATE TABLE PropertyForRent (
propertyNo      PNumber        NOT NULL, ….
rooms             PRooms              NOT NULL       DEFAULT 4,
rent                PRent                      NOT NULL,      DEFAULT 600,
ownerNo                OwnerNumber           NOT NULL,
staffNo                StaffNumber
            Constraint StaffNotHandlingTooMuch ….
branchNo               BranchNumber          NOT NULL,
PRIMARY KEY (propertyNo),
FOREIGN KEY (staffNo) REFERENCES Staff
        ON DELETE SET NULL ON UPDATE CASCADE ….);
```

ALTER TABLE

- Add a new column to a table.
- Drop a column from a table.
- Add a new table constraint.
- Drop a table constraint.
- Set a default for a column.
- Drop a default for a column.

Example: ALTER TABLE

- Change Staff table by removing default of ‘Assistant’ for position column and setting default for sex column to female (‘F’).

```
ALTER TABLE Staff
ALTER position DROP DEFAULT;
ALTER TABLE Staff
ALTER sex SET DEFAULT ‘F’;
```

Example: ALTER TABLE

- Remove constraint from PropertyForRent that staff are not allowed to handle more than 100 properties at a time. Add new column to Client table.

```
ALTER TABLE PropertyForRent
        DROP CONSTRAINT StaffNotHandlingTooMuch;
ALTER TABLE Client
        ADD prefNoRooms PRooms;
```

DROP TABLE

- DROP TABLE TableName [RESTRICT | CASCADE]

```
e.g.     DROP TABLE PropertyForRent;
```

- Removes named table and all rows within it.
- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).

Views

- View
  - Dynamic result of one or more relational operations operating on base relations to produce another relation.

- Virtual relation that does not necessarily actually exist in the database but is produced upon request, at time of request.

Views

- Contents of a view are defined as a query on one or more base relations.
- With view resolution, any operations on view are automatically translated into operations on relations from which it is derived.
- With view materialization, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.

SQL - CREATE VIEW

```
CREATE VIEW ViewName [ (newColumnName [,...]) ]
AS subselect
[WITH [CASCADED | LOCAL] CHECK OPTION]
```

- Can assign a name to each column in view.
- If list of column names is specified, it must have same number of items as number of columns produced by subselect.
- If omitted, each column takes name of corresponding column in subselect.

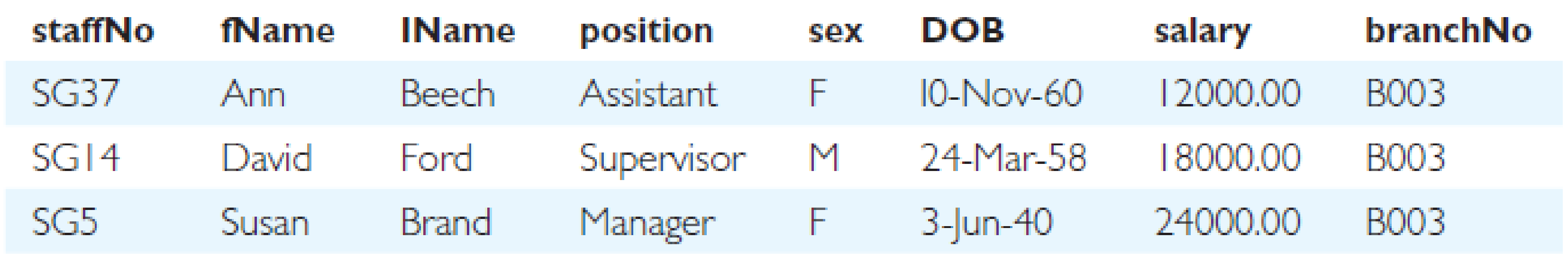
SQL - CREATE VIEW

- List must be specified if there is any ambiguity in a column name.
- The subselect is known as the defining query.
- WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.
- Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.

Example: Create Horizontal View

- Create view so that manager at branch B003 can only see details for staff who work in his or her office.

```
CREATE VIEW Manager3Staff
AS      SELECT *
        FROM Staff
        WHERE branchNo = ‘B003’;
```



staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

Example: Create Vertical View

- Create view of staff details at branch B003 excluding salaries.

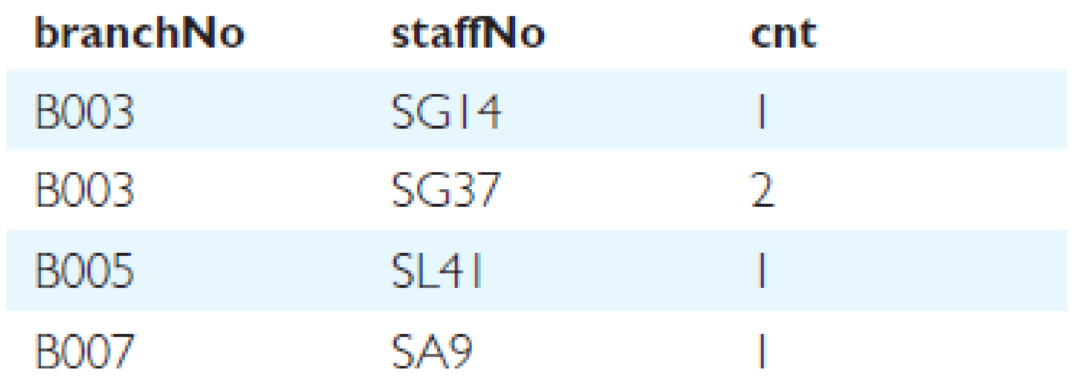
```
CREATE VIEW Staff3
AS SELECT staffNo, fName, lName, position, sex
FROM Staff
WHERE branchNo = 'B003';
```

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

Example: Grouped and Joined Views

- Create view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
 AS SELECT s.branchNo, s.staffNo, COUNT(*)
       FROM Staff s, PropertyForRent p
       WHERE s.staffNo = p.staffNo
       GROUP BY s.branchNo, s.staffNo;
```



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

SQL - DROP VIEW

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

- Causes definition of view to be deleted from database.
- For example:

```
DROP VIEW Manager3Staff;
```

- With CASCADE, all related dependent objects are deleted; i.e. any views defined on view being dropped.
- With RESTRICT (default), if any other objects depend for their existence on continued existence of view being dropped, command is rejected

View Resolution

- Count number of properties managed by each member at branch B003.

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = ‘B003’
ORDER BY staffNo;
```

View Resolution

a) View column names in SELECT list are translated into their corresponding column names in the defining query:

```
SELECT s.staffNo As staffNo, COUNT(*) As cnt
```

b) View names in FROM are replaced with corresponding FROM lists of defining query:

```
FROM Staff s, PropertyForRent p
```

c) WHERE from user query is combined with WHERE of defining query using AND:

```
WHERE s.staffNo = p.staffNo AND branchNo = ‘B003’
```

d) GROUP BY and HAVING clauses copied from defining query:

```
GROUP BY s.branchNo, s.staffNo
```

View Resolution

e) ORDER BY copied from query with view column name translated into defining query column name

```
ORDER BY s.staffNo
```

f) Final merged query is now executed to produce the result:

```
SELECT s.staffNo AS staffNo, COUNT(*) AS cnt
FROM Staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo AND
        branchNo = ‘B003’
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```

Restrictions on Views

- SQL imposes several restrictions on creation and use of views.

  1. If column in view is based on an aggregate function:
     - Column may appear only in SELECT and ORDER BY clauses of queries that access view.
     - Column may not be used in WHERE nor be an argument to an aggregate function in any query based on view.

  - For example, following query would fail:

```
SELECT COUNT(cnt)
FROM StaffPropCnt;
```

  - Similarly, following query would also fail:

```
SELECT *
FROM StaffPropCnt
WHERE cnt > 2;
```

Restrictions on Views

2. Grouped view may never be joined with a base table or a view.

- For example, StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.

View Updatability

- All updates to base table reflected in all views that encompass base table.
- Similarly, may expect that if view is updated then base table(s) will reflect change.

- However, consider again view StaffPropCnt.
- If we tried to insert record showing that at branch B003, SG5 manages 2 properties:

```
INSERT INTO StaffPropCnt
VALUES ('B003', 'SG5', 2);
```

- Have to insert 2 records into PropertyForRent showing which properties SG5 manages. However, do not know which properties they are; i.e. do not know primary keys!

View Updatability

- If change definition of view and replace count with actual property numbers:

```
CREATE VIEW StaffPropList (branchNo,  staffNo, propertyNo)
AS SELECT s.branchNo, s.staffNo, p.propertyNo
            FROM Staff s, PropertyForRent p
            WHERE s.staffNo = p.staffNo;
```

- Now try to insert the record:

```
INSERT INTO StaffPropList
VALUES (‘B003’, ‘SG5’, ‘PG19’);
```

- Still problem, because in PropertyForRent all columns except postcode/staffNo are not allowed nulls.
- However, have no way of giving remaining non-null columns values.

View Updatability

- ISO specifies that a view is updatable if and only if:
  - DISTINCT is not specified.
  - Every element in SELECT list of defining query is a column name and no column appears more than once.
  - FROM clause specifies only one table, excluding any views based on a join, union, intersection or difference.
  - No nested SELECT referencing outer table.
  - No GROUP BY or HAVING clause.
  - Also, every row added through view must not violate integrity constraints of base table.

Updatable View

- For view to be updatable, DBMS must be able to trace any row or column back to its row or column in the source table.

WITH CHECK OPTION

- Rows exist in a view because they satisfy WHERE condition of defining query.
- If a row changes and no longer satisfies condition, it disappears from the view.
- New rows appear within view when insert/update on view cause them to satisfy WHERE condition.
- Rows that enter or leave a view are called migrating rows.
- WITH CHECK OPTION prohibits a row migrating out of the view.
- LOCAL/CASCADED apply to view hierarchies.
- With LOCAL, any row insert/update on view and any view directly or indirectly defined on this view must not cause row to disappear from view unless row also disappears from derived view/table.
- With CASCADED (default), any row insert/ update on this view and on any view directly or indirectly defined on this view must not cause row to disappear from the view.

Example: WITH CHECK OPTION

```
CREATE VIEW Manager3Staff
AS        SELECT *
          FROM Staff
          WHERE branchNo = ‘B003’
WITH CHECK OPTION;
```

- Cannot update branch number of row B003 to B002 as this would cause row to migrate from view.
- Also cannot insert a row into view with a branch number that does not equal B003.

Example: WITH CHECK OPTION

- Now consider the following:

```
CREATE VIEW LowSalary
        AS      SELECT * FROM Staff WHERE salary > 9000;
CREATE VIEW HighSalary
        AS      SELECT * FROM LowSalary
        WHERE salary > 10000
        WITH LOCAL CHECK OPTION;
CREATE VIEW Manager3Staff
        AS      SELECT * FROM HighSalary
        WHERE branchNo = ‘B003’;
```

Example: WITH CHECK OPTION

```
UPDATE Manager3Staff
SET salary = 9500
WHERE staffNo = 'SG37';
```

- This update would fail: although update would cause row to disappear from HighSalary, row would not disappear from LowSalary.
- However, if update tried to set salary to 8000, update would succeed as row would no longer be part of LowSalary.

Example 7.6 - WITH CHECK OPTION

- If HighSalary had specified WITH CASCADED CHECK OPTION, setting salary to 9500 or 8000 would be rejected because row would disappear from HighSalary.
- To prevent anomalies like this, each view should be created using WITH CASCADED CHECK OPTION.

Pearson Education © 2014

Advantages of Views

- Data independence
- Currency
- Improved security
- Reduced complexity
- Convenience
- Customization
- Data integrity

Disadvantages of Views

- Update restriction
- Structure restriction
- Performance

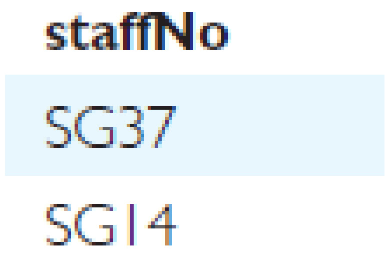
View Materialization

- View resolution mechanism may be slow, particularly if view is accessed frequently.
- View materialization stores view as temporary table when view is first queried.
- Thereafter, queries based on materialized view can be faster than recomputing view each time.
- Difficulty is maintaining the currency of view while base tables(s) are being updated.

View Maintenance

- View maintenance aims to apply only those changes necessary to keep view current.
- Consider following view:

```
CREATE VIEW StaffPropRent(staffNo)
AS        SELECT DISTINCT staffNo
          FROM PropertyForRent
          WHERE branchNo = 'B003' AND
                    rent > 400;
```



staffNo
SG37
SG14

View Materialization

- If insert row into PropertyForRent with rent ≤400 then view would be unchanged.
- If insert row for property PG24 at branch B003 with staffNo = SG19 and rent = 550, then row would appear in materialized view.
- If insert row for property PG54 at branch B003 with staffNo = SG37 and rent = 450, then no new row would need to be added to materialized view.
- If delete property PG24, row should be deleted from materialized view.
- If delete property PG54, then row for PG37 should not be deleted (because of existing property PG21).

Transactions

- SQL defines transaction model based on COMMIT and ROLLBACK.
- Transaction is logical unit of work with one or more SQL statements guaranteed to be atomic with respect to recovery.
- An SQL transaction automatically begins with a transaction-initiating SQL statement (e.g., SELECT, INSERT).
- Changes made by transaction are not visible to other concurrently executing transactions until transaction completes.

Transactions

- Transaction can complete in one of four ways:
  - COMMIT ends transaction successfully, making changes permanent.
  - ROLLBACK aborts transaction, backing out any changes made by transaction.
  - For programmatic SQL, successful program termination ends final transaction successfully, even if COMMIT has not been executed.
  - For programmatic SQL, abnormal program end aborts transaction.

Transactions

- New transaction starts with next transaction-initiating statement.
- SQL transactions cannot be nested.
- SET TRANSACTION configures transaction:

```
SET TRANSACTION
[READ ONLY | READ WRITE] |
[ISOLATION LEVEL READ UNCOMMITTED |
READ COMMITTED|REPEATABLE READ |SERIALIZABLE ]
```

Immediate and Deferred Integrity Constraints

- Do not always want constraints to be checked immediately, but instead at transaction commit.
- Constraint may be defined as INITIALLY IMMEDIATE or INITIALLY DEFERRED, indicating mode the constraint assumes at start of each transaction.
- In former case, also possible to specify whether mode can be changed subsequently using qualifier [NOT] DEFERRABLE.
- Default mode is INITIALLY IMMEDIATE.

Immediate and Deferred Integrity Constraints

- SET CONSTRAINTS statement used to set mode for specified constraints for current transaction:

```
SET CONSTRAINTS
{ALL | constraintName [, . . . ]}
        {DEFERRED ¦ IMMEDIATE}
```

Access Control - Authorization Identifiers and Ownership

- Authorization identifier is normal SQL identifier used to establish identity of a user. Usually has an associated password.
- Used to determine which objects user may reference and what operations may be performed on those objects.
- Each object created in SQL has an owner, as defined in AUTHORIZATION clause of schema to which object belongs.
- Owner is only person who may know about it.

Privileges

- Actions user permitted to carry out on given base table or view:

SELECT	Retrieve data from a table.
INSERT	Insert new rows into a table.
UPDATE	Modify rows of data in a table.
DELETE	Delete rows of data from a table.
REFERENCES	Reference columns of named table in integrity constraints.
USAGE	Use domains, collations, character sets, and translations.

- Can restrict INSERT/UPDATE/REFERENCES to named columns.
- Owner of table must grant other users the necessary privileges using GRANT statement.
- To create view, user must have SELECT privilege on all tables that make up view and REFERENCES privilege on the named columns.

GRANT

```
GRANT       {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
TO {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

- PrivilegeList consists of one or more of above privileges separated by commas.
- ALL PRIVILEGES grants all privileges to a user.

- PUBLIC allows access to be granted to all present and future authorized users.
- ObjectName can be a base table, view, domain, character set, collation or translation.
- WITH GRANT OPTION allows privileges to be passed on.

Example: GRANT

- Give Manager full privileges to Staff table.

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager WITH GRANT OPTION;
```

- Give users Personnel and Director SELECT and UPDATE on column salary of Staff.

```
GRANT SELECT, UPDATE (salary)
ON Staff
TO Personnel, Director;
```

Example: GRANT Specific Privileges to PUBLIC

- Give all users SELECT on Branch table.

```
GRANT SELECT
ON Branch
TO PUBLIC;
```

REVOKE

- REVOKE takes away privileges granted with GRANT.

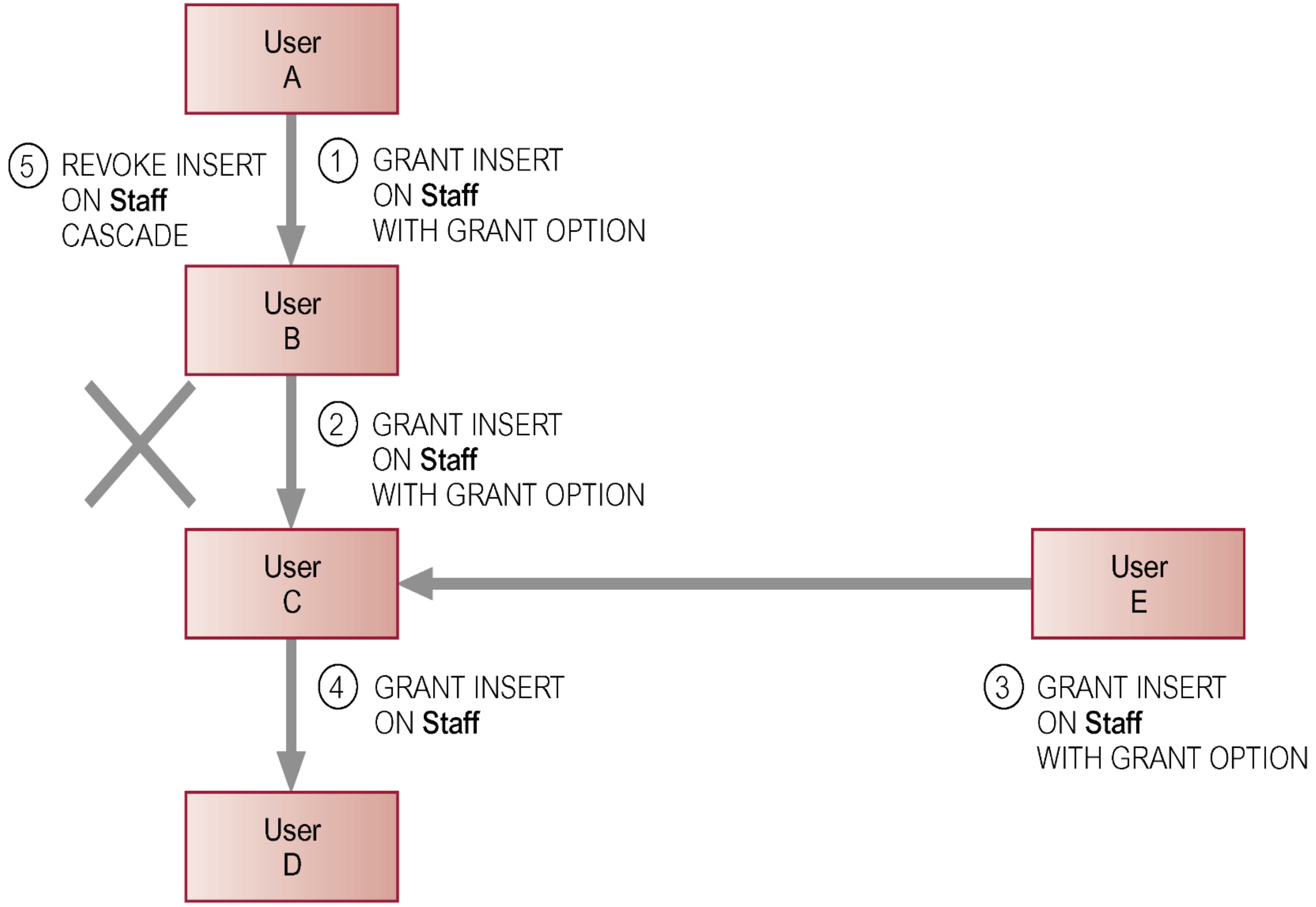
```
REVOKE [GRANT OPTION FOR]
            {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
              [RESTRICT | CASCADE]
```

- ALL PRIVILEGES refers to all privileges granted to a user by user revoking privileges.

REVOKE

- GRANT OPTION FOR allows privileges passed on via WITH GRANT OPTION of GRANT to be revoked separately from the privileges themselves.
- REVOKE fails if it results in an abandoned object, such as a view, unless the CASCADE keyword has been specified.
- Privileges granted to this user by other users are not affected.

REVOKE


User A
5 REVOKE INSERT ON Staff CASCADE
1 GRANT INSERT ON Staff WITH GRANT OPTION
User B
2 GRANT INSERT ON Staff WITH GRANT OPTION
User C
User E
4 GRANT INSERT ON Staff
3 GRANT INSERT ON Staff WITH GRANT OPTION
User D

Example: REVOKE Specific Privileges

- Revoke privilege SELECT on Branch table from all users.

```
REVOKE SELECT
ON Branch
FROM PUBLIC;
```

- Revoke all privileges given to Director on Staff table.

```
REVOKE ALL PRIVILEGES
ON Staff
FROM Director;
```

Advanced SQL

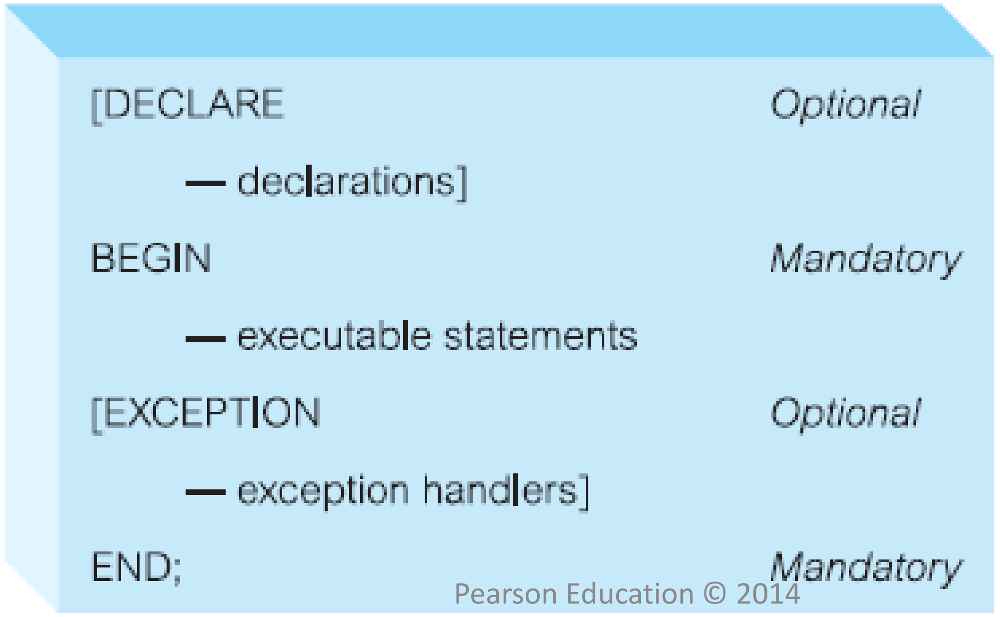
The SQL Programming Language

- Impedance mismatch
  - Mixing different programming paradigms
  - SQL is a declarative language
  - High-level language such as C is a procedural language
  - SQL and 3GLs use different models to represent data

- SQL/PSM (Persistent Stored Modules)
- PL/SQL (Procedural Language/SQL)
  - Oracle’s procedural extension to SQL
  - Two versions

Declarations

- Variables and constant variables must be declared before they can be referenced
- Possible to declare a variable as NOT NULL
- %TYPE – variable same type as a column
  - vStaffNo   Staff.staffNo%TYPE;
- %ROWTYPE – variable same type as an entire row
  - vStaffNo1   Staff%ROWTYPE;



```
[DECLARE                          Optional
    — declarations]
BEGIN                             Mandatory
    — executable statements
[EXCEPTION                        Optional
    — exception handlers]
END;                              Mandatory
```

Assignments

- Variables can be assigned in two ways:
  - Using the normal assignment statement (:=):

```
vStaffNo := 'SG14';
```

  - Using an SQL SELECT or FETCH statement:

```
SELECT COUNT(*) INTO x
FROM PropertyForRent
WHERE staffNo = vStaffNo;
```

Control Statements

- Conditional IF statement
- Conditional CASE statement
- Iteration statement (LOOP)
- Iteration statement (WHILE and REPEAT)
- Iteration statement (FOR)

Conditional IF Statement

```
IF (position = ‘Manager’) THEN
        salary := salary*1.05;
ELSE
        salary := salary*1.05;
END IF;
```

Conditional CASE Statement

```
UPDATE Staff
SET salary  =  CASE
          WHEN position  =  ‘Manager’
          THEN salary * 1.05
          ELSE
          salary * 1.02
END;
```

Iteration Statement (LOOP)

```
x: = 1;
myLoop:
LOOP
        x := x + 1;
        IF (x > 3) THEN
                EXIT myLoop;        --- exit loop now
END LOOP myLoop;
--- control resumes here
y := 2;
```

Iteration Statement (WHILE and REPEAT)

```
WHILE (condition) DO
        <SQL statement list>
END WHILE [labelName];

REPEAT
        <SQL statement list>
UNTIL (condition)
END REPEAT [labelName];
```

Iteration Statement (FOR)

```
myLoop1:
FOR iStaff AS SELECT COUNT(*) FROM PropertyForRent WHERE staffNo =
'SG14' DO
                    .....
END FOR myLoop1;
```

Exceptions in PL/SQL

- Exception
  - Identifier in PL/SQL
  - Raised during the execution of a block
  - Terminates block's main body of actions
- Exception handlers
  - Separate routines that handle raised exceptions
- User-defined exception
  - Defined in the declarative part of a PL/SQL block

Example of Exception Handling in PL/SQL

```
DECLARE
   vpCount      NUMBER;
   vStaffNo PropertyForRent.staffNo%TYPE := 'SG14';
-- define an exception for the enterprise constraint that prevents a member of staff
-- managing more than 100 properties
   e_too_many_properties EXCEPTION;
   PRAGMA EXCEPTION_INIT(e_too_many_properties, -20000);
BEGIN
          SELECT COUNT(*) INTO vpCount
          FROM PropertyForRent
          WHERE staffNo = vStaffNo;
          IF vpCount  = 100
-- raise an exception for the general constraint
               RAISE e_too_many_properties;
          END IF;
          UPDATE PropertyForRent SET staffNo = vStaffNo WHERE propertyNo = 'PG4';
EXCEPTION
   -- handle the exception for the general constraint
   WHEN e_too_many_properties THEN
               dbms_output.put_line('Member of staff ' || staffNo || 'already managing 100 properties');
END;
```

Condition Handling

- Define a handler by:
  - Specifying its type
  - Exception and completion conditions it can resolve
  - Action it takes to do so
- Handler is activated:
  - When it is the most appropriate handler for the condition that has been raised by the SQL statement

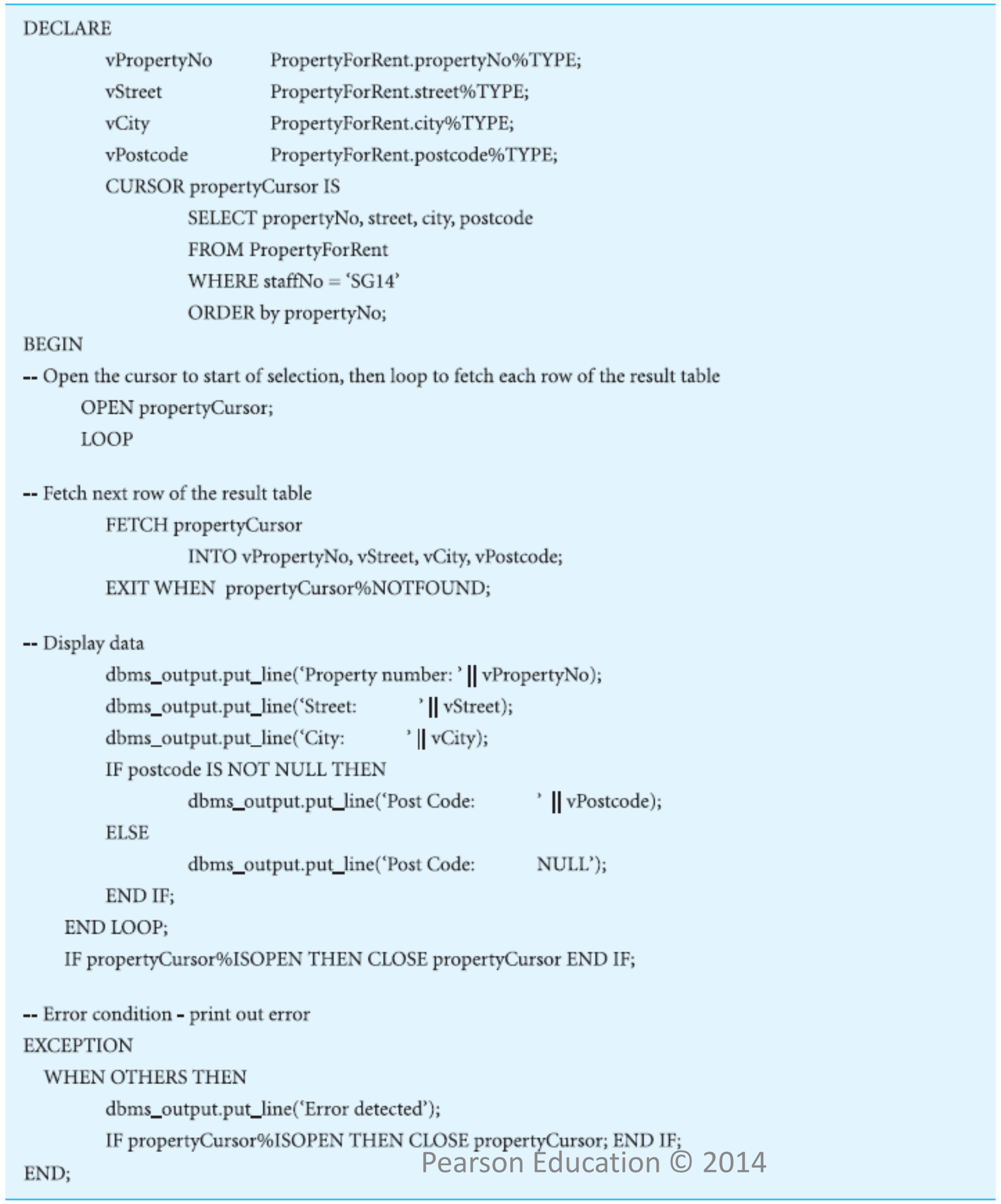
The DECLARE . . . HANDLER Statement

- **DECLARE {CONTINUE | EXIT | UNDO} HANDLER FOR SQLSTATE** {sqlstateValue | conditionName | **SQLEXCEPTION |SQLWARNING | NOT FOUND}** handlerAction;

Cursors in PL/SQL

- Cursor
  - Allows the rows of a query result to be accessed one at a time
  - Must be declared and opened before use
  - Must be closed to deactivate it after it is no longer required
  - Updating rows through a cursor

Using Cursors in PL/SQL to Process a Multirow Query



```
DECLARE
        vPropertyNo        PropertyForRent.propertyNo%TYPE;
        vStreet            PropertyForRent.street%TYPE;
        vCity              PropertyForRent.city%TYPE;
        vPostcode          PropertyForRent.postcode%TYPE;
        CURSOR propertyCursor IS
                SELECT propertyNo, street, city, postcode
                FROM PropertyForRent
                WHERE staffNo = 'SG14'
                ORDER by propertyNo;
BEGIN
-- Open the cursor to start of selection, then loop to fetch each row of the result table
      OPEN propertyCursor;
      LOOP

-- Fetch next row of the result table
        FETCH propertyCursor
                INTO vPropertyNo, vStreet, vCity, vPostcode;
        EXIT WHEN  propertyCursor%NOTFOUND;

-- Display data
        dbms_output.put_line('Property number: ' || vPropertyNo);
        dbms_output.put_line('Street:          ' || vStreet);
        dbms_output.put_line('City:            ' || vCity);
        IF postcode IS NOT NULL THEN
                dbms_output.put_line('Post Code:          ' || vPostcode);
        ELSE
                dbms_output.put_line('Post Code:          NULL');
        END IF;
    END LOOP;
    IF propertyCursor%ISOPEN THEN CLOSE propertyCursor END IF;

-- Error condition - print out error
EXCEPTION
  WHEN OTHERS THEN
        dbms_output.put_line('Error detected');
        IF propertyCursor%ISOPEN THEN CLOSE propertyCursor; END IF;
END;
```

Subprograms, Stored Procedures, Functions, and Packages

- Subprograms
  - Named PL/SQL blocks that can take parameters and be invoked
- Two types:
  - Stored procedures
  - Functions (returns a single value to caller)
- Can take a set of parameters
  - Each has name and data type
  - Can be designated as IN, OUT, IN OUT

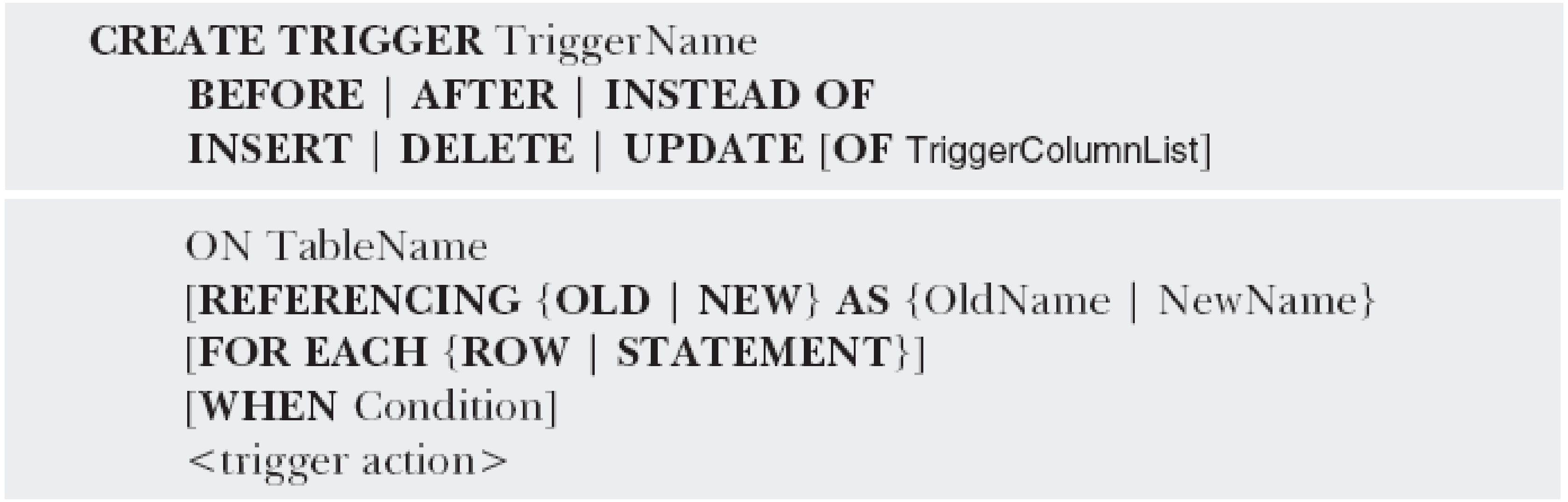
Subprograms, Stored Procedures, Functions, and Packages

- Package
  - Collection of procedures, functions, variables, and SQL statements that are grouped together and stored as a single program unit
- Specification
  - Declares all public constructs of the package
- Body
  - Defines all constructs (public and private) of the package

Triggers

- Trigger
  - Defines an action that the database should take when some event occurs in the application
  - Based on Event-Condition-Action (ECA) model
- Types
  - Row-level
  - Statement-level
- Event: INSERT, UPDATE or DELETE
- Timing: BEFORE, AFTER or INSTEAD OF
- Advantages and disadvantages of triggers

Trigger Format



```
CREATE TRIGGER TriggerName
        BEFORE | AFTER | INSTEAD OF
        INSERT | DELETE | UPDATE [OF TriggerColumnList]
        ON TableName
        [REFERENCING {OLD | NEW} AS {OldName | NewName}
        [FOR EACH {ROW | STATEMENT}]
        [WHEN Condition]
        <trigger action>
```

Using a BEFORE Trigger

```
CREATE TRIGGER StaffNotHandlingTooMuch
BEFORE INSERT ON PropertyForRent
REFERENCING NEW AS newrow
FOR EACH ROW
DECLARE
  vpCount          NUMBER;
BEGIN
          SELECT COUNT(*) INTO vpCount
          FROM PropertyForRent
          WHERE staffNo = :newrow.staffNo;
          IF vpCount = 100
             raise_application_error(-20000, ('Member' || :newrow.staffNo || 'already managing 100 properties');
          END IF;
END;
```

Triggers – Advantages

- Elimination of redundant code
- Simplifying modifications
- Increased security
- Improved integrity
- Improved processing power
- Good fit with client-server architecture

Triggers – Disadvantages

- Performance overhead
- Cascading effects
- Cannot be scheduled
- Less portable

Recursion

- Extremely difficult to handle recursive queries
  - Queries about relationships that a relation has with itself (directly or indirectly)
- WITH RECURSIVE statement handles this
- Infinite loop can occur unless the cycle can be detected
  - CYCLE clause

Recursion - Example

```
WITH RECURSIVE
AllManagers (staffNo, managerStaffNo) AS
(SELECT staffNo, managerStaffNo
FROM Staff
UNION
SELECT in.staffNo, out.managerStaffNo
FROM AllManagers in, Staff out
WHERE in.managerStaffNo = out.staffNo);
SELECT * FROM AllManagers
ORDER BY staffNo, managerStaffNo;
```