Database Systems

Bilal Khalid Dar

Agenda

- Purpose and importance of SQL.
- How to retrieve data from database using SELECT

and:

  - Use compound WHERE conditions.
  - Sort query results using ORDER BY.
  - Use aggregate functions.

Objectives of SQL

Ideally, database language should allow user to:
– create the database and relation structures;

– perform insertion, modification, deletion of data from relations;

– perform simple and complex queries.

Must perform these tasks with minimal user effort and command structure/syntax must be easy to learn.

It must be portable.

Objectives of SQL

SQL is a transform-oriented language with 2 major components:

– A DDL for defining database structure.

– A DML for retrieving and updating data.

Until 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.

Objectives of SQL

SQL is relatively easy to learn:

– it is non-procedural - you specify *what* information you require, rather than *how* to get it;

– it is essentially free-format

SQL Data Definition and Data Types

- Terminology:
  - **Table**, **row**, and **column** used for relational model terms relation, tuple, and attribute
- `CREATE` statement
  - Main SQL command for data definition

Attribute Data Types and Domains in SQL

- Basic **data types**
  - **Numeric** data types
    - Integer numbers: `INTEGER`, `INT`, and `SMALLINT`
    - Floating-point (real) numbers: `FLOAT` or `REAL`, and `DOUBLE PRECISION`
  - **Character-string** data types
    - Fixed length: `CHAR(`*n*`)`, `CHARACTER(`*n*`)`
    - Varying length : `VARCHAR(`*n*`)`, `CHAR VARYING(`*n*`)`, `CHARACTER VARYING(`*n*`)`

Attribute Data Types and Domains in SQL (cont'd.)

- **Bit-string** data types
  - Fixed length: `BIT(`*n*`)`
  - Varying length: `BIT VARYING(`*n*`)`
- **Boolean** data type
  - Values of `TRUE` or `FALSE` or `NULL`
- **DATE** data type
  - Ten positions
  - Components are `YEAR`, `MONTH`, and `DAY` in the form YYYY-MM-DD

Attribute Data Types and Domains in SQL (cont'd.)

- Additional data types
  - **Timestamp** data type (`TIMESTAMP`)
    - Includes the `DATE` and `TIME` fields
    - Plus a minimum of six positions for decimal fractions of seconds
    - Optional `WITH TIME ZONE` qualifier
  - **`INTERVAL`** data type
    - Specifies a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp

Attribute Data Types and Domains in SQL (cont'd.)

- Domain
  - Name used with the attribute specification
  - Makes it easier to change the data type for a domain that is used by numerous attributes
  - Improves schema readability
  - Example:
    - `CREATE DOMAIN SSN_TYPE AS CHAR(9);`

Specifying Constraints in SQL

- Basic constraints:
  - Key and referential integrity constraints
  - Restrictions on attribute domains and NULLs
  - Constraints on individual tuples within a relation

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;

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.

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

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.

History of SQL

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.

Importance of SQL

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 *userdefined words*.

– **Reserved** words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.

– **User-defined** words are made up by user and represent names of various database objects such as relations, columns, views.

Writing SQL Commands

Most components of an SQL statement are *case insensitive*, except for literal character data.

More readable with indentation and lineation:

– Each clause should begin on a new line.

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

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

Writing SQL Commands

Use extended form of BNF notation:

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

Literals

Literals are constants used in SQL statements.

All non-numeric literals must be enclosed in
single quotes (e.g. ‘London’).

All numeric literals must not be enclosed in quotes (e.g. 650.00)

SELECT STATEMENT

SELECT Statement

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

SELECT Statement

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

SELECT Statement

Order of the clauses cannot be changed.

Only SELECT and FROM are mandatory.

Example - All Columns, All Rows

List full details of all staff.

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

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

```
SELECT *
FROM Staff;
```

Example - All Columns, All Rows

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

Example - All Columns, All Rows

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

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

Example - All Columns, All Rows

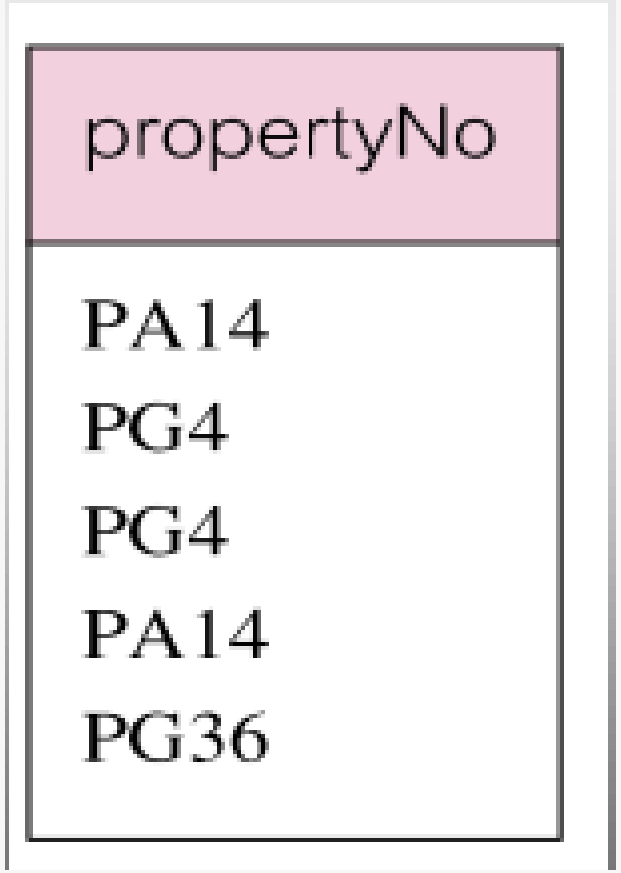
staffNo	fName	lName	salary
SL21	John	White	30000.00
SG37	Ann	Beech	12000.00
SG14	David	Ford	18000.00
SA9	Mary	Howe	9000.00
SG5	Susan	Brand	24000.00
SL41	Julie	Lee	9000.00

Example - Use of DISTINCT

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

- SELECT propertyNo

FROM Viewing;



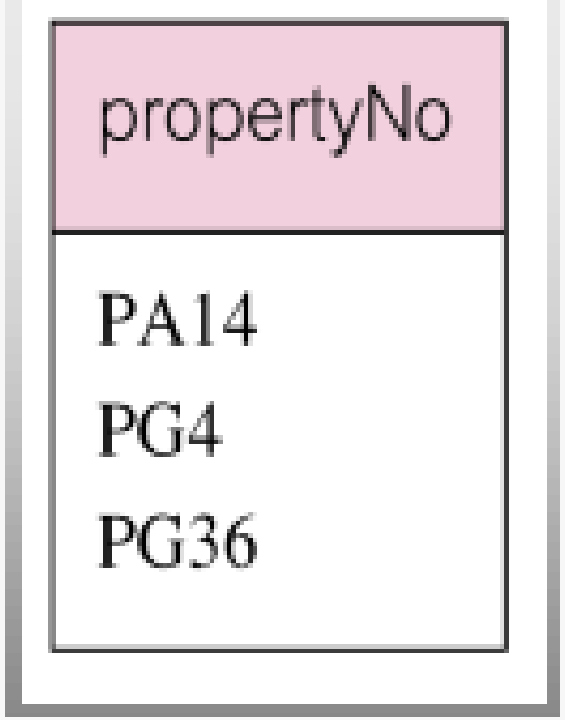
propertyNo
PA14
PG4
PG4
PA14
PG36

Example - Use of DISTINCT

- Use DISTINCT to eliminate duplicates:

- SELECT DISTINCT propertyNo

FROM Viewing;



propertyNo
PA14
PG4
PG36

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

Example Calculated Fields

- To name column, use AS clause:
- SELECT staffNo, fName, lName, salary/12

AS monthlySalary

FROM Staff;

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

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

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

Example Range Search Condition

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

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.

Example Set Membership

List all managers and supervisors.

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

Table 5.8 Result table for Example 5.8.

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

Example Set Membership

- 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 more efficient when set contains many values

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%';
```

Table 5.9 Result table for Example 5.9.

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

Example Pattern Matching

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

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;
```

Example NULL Search Condition

clientNo	viewDate
CR56	26-May-04

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

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;
```

Example Single Column Ordering

Table 5.11 Result table for Example 5.11.

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

Example Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

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

Example Multiple Column Ordering

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

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

Example Multiple Column Ordering

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

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

Example Multiple Column Ordering

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

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

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 single column of table and returns single value
- COUNT, MIN, and MAX apply to numeric and non-numeric fields
  - SUM and AVG used on numeric fields only
- Each function eliminates nulls first and operates only on remaining non-null values
  - Except COUNT

SELECT Statement - Aggregates

- COUNT(*) counts all rows of table
  - Includes nulls and duplicate values
- Can use DISTINCT before column name to eliminate duplicates
- DISTINCT has no effect with MIN/MAX
  - Has effect with SUM/AVG

SELECT Statement - Aggregates

- Aggregate functions used only in SELECT list and HAVING clause

- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference column out with aggregate function
- Illegal:

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

Example Use of COUNT(*)

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

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

myCount
5

Example Use of COUNT(DISTINCT)

How many different properties viewed in May ‘04?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN ‘1-May-0
      AND ‘31-May-04’;
```

myCount
2

Example Use of COUNT and SUM

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

Example Use of MIN, MAX, AVG

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

Agenda

- How to retrieve data from database using SELECT

and:

  - Group data using GROUP BY and HAVING.
  - Use sub queries.
  - Create, Insert, Delete, Update

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***
  - **SELECT clause may only contain:**
    - **column names**
    - **aggregate functions**
    - **constants**
    - **expression involving combinations of above**

SELECT Statement - Grouping

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

Example Use of GROUP BY

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;
```

Example Use of GROUP BY

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00
B007	1	9000.00

Restricted Groupings – HAVING clause

- HAVING clause designed for use with GROUP BY to restrict groups that appear in final result table
- Similar to WHERE:
  - WHERE filters individual rows
  - HAVING filters groups
- Column names in HAVING clause must appear in GROUP BY list or be contained within aggregate function

Example Use of HAVING

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

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

Example Use of HAVING

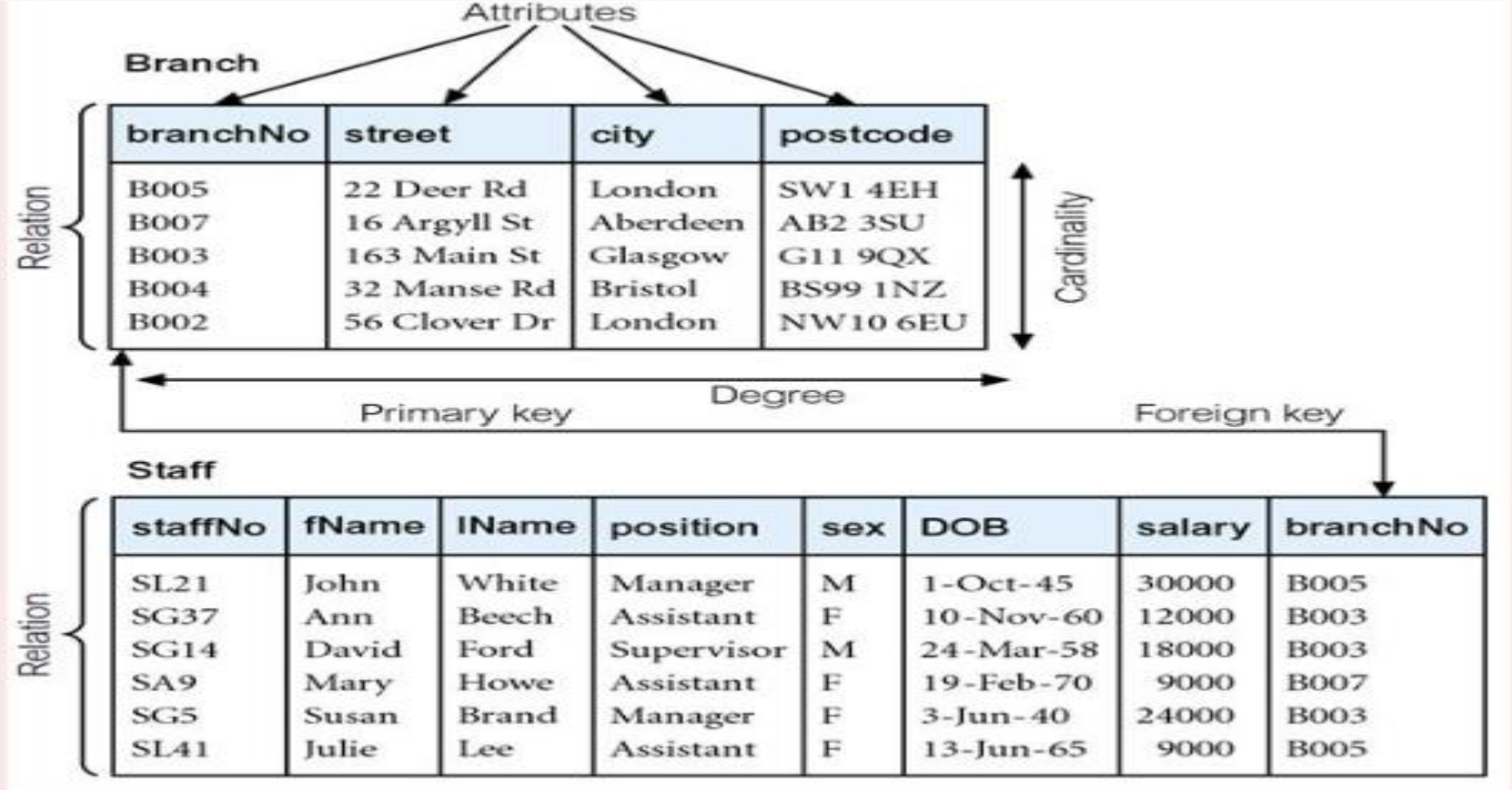
branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Subqueries

- **Some SQL statements can have SELECT embedded within them**
- **Ssubselect can be used in WHERE and HAVING clauses of an outer SELECT**
  - **Called *subquery* or *nested query***
- **Subselects may also appear in INSERT, UPDATE, and DELETE statements**

Example Subquery with Equality

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



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

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

Example Subquery with Equality

List staff who work in branch at ‘163 Main St’.

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

Example Subquery with Equality

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

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

Example Subquery with Equality

Table 5.19 Result table for Example 5.19.

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

Example Subquery with Aggregate

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);
```

Example Subquery with Aggregate

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

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

Example Subquery with Aggregate

Table 5.20 Result table for Example 5.20.

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

Subquery Rules

- ORDER BY clause may not be used in subquery
  - May be used in outermost SELECT
- Subquery SELECT list must consist of 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 table in FROM using *alias*

Subquery Rules

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

Example Nested subquery: use of IN

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’));
```

ANY and ALL

- ANY and ALL used with subqueries that produce single column of numbers
- ALL
  - Condition only true if satisfied by *all* values produced by subquery
- ANY
  - Condition true if satisfied by *any* values produced by subquery
- If subquery empty
  - ALL returns true
  - ANY returns false
- SOME may be used in place of ANY

Example 6.22 Use of ANY/SOME

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');
```

Example 6.22 Use of ANY/SOME

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

Table 5.22 Result table for Example 5.22.

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

Example 6.23 Use of ALL

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');
```

Example 6.23 Use of ALL

Table 5.23 Result table for Example 5.23.

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

The CREATE TABLE Command in SQL

  - Provide name
  - Specify attributes and initial constraints
- Can optionally specify schema:
  - `CREATE TABLE COMPANY.EMPLOYEE ...`

    or
  - `CREATE TABLE EMPLOYEE ...`

```
CREATE TABLE EMPLOYEE
        ( Fname             VARCHAR(15)          NOT NULL,
          Minit             CHAR,
          Lname             VARCHAR(15)          NOT NULL,
          Ssn               CHAR(9)              NOT NULL,
          Bdate             DATE,
          Address           VARCHAR(30),
          Sex               CHAR,
          Salary            DECIMAL(10,2),
          Super_ssn         CHAR(9),
          Dno               INT                  NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE DEPARTMENT
        ( Dname             VARCHAR(15)          NOT NULL,
          Dnumber           INT                  NOT NULL,
          Mgr_ssn           CHAR(9)              NOT NULL,
          Mgr_start_date    DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

Figure 4.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 3.7.

```
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber           INT             NOT NULL,
          Dlocation         VARCHAR(15)     NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname             VARCHAR(15)     NOT NULL,
          Pnumber           INT             NOT NULL,
          Plocation         VARCHAR(15),
          Dnum              INT             NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE WORKS_ON
        ( Essn              CHAR(9)         NOT NULL,
          Pno               INT             NOT NULL,
          Hours             DECIMAL(3,1)    NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn              CHAR(9)         NOT NULL,
          Dependent_name    VARCHAR(15)     NOT NULL,
          Sex               CHAR,
          Bdate             DATE,
          Relationship      VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Figure 4.1
SQL CREATE TABLE data definition statements for defining the COMPANY schema from Figure 3.7.

The CREATE TABLE Command in SQL (cont'd.)

- Some foreign keys may cause errors
  - Specified either via:
    - Circular references
    - Or because they refer to a table that has not yet been created

Specifying Constraints in SQL

- Basic constraints:
  - Key and referential integrity constraints
  - Restrictions on attribute domains and NULLs
  - Constraints on individual tuples within a relation

Specifying Attribute Constraints and Attribute Defaults

- `NOT NULL`
  - `NULL` is not permitted for a particular attribute
- Default value
  - **`DEFAULT`** `<value>`
- **`CHECK`** clause
  - `Dnumber INT NOT NULL CHECK (Dnumber > 0 AND Dnumber < 21);`

```
CREATE TABLE DEPARTMENT
    ( . . . ,
      Mgr_ssn     CHAR(9)        NOT NULL          DEFAULT '888665555',
      . . . ,
    CONSTRAINT DEPTPK
      PRIMARY KEY(Dnumber),
    CONSTRAINT DEPTSK
      UNIQUE (Dname),
    CONSTRAINT DEPTMGRFK
      FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn)
                  ON DELETE SET DEFAULT   ON UPDATE CASCADE);
CREATE TABLE DEPT_LOCATIONS
    ( . . . ,
    PRIMARY KEY (Dnumber, Dlocation),
    FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber)
                  ON DELETE CASCADE          ON UPDATE CASCADE);
```

Figure 4.2
Example illustrating how default attribute values and referential integrity triggered actions are specified in SQL.

Specifying Key and Referential Integrity Constraints

- **`PRIMARY KEY`** clause
  - Specifies one or more attributes that make up the primary key of a relation
  - `Dnumber INT PRIMARY KEY;`
- **`UNIQUE`** clause
  - Specifies alternate (secondary) keys
  - `Dname VARCHAR(15) UNIQUE;`

Specifying Key and Referential Integrity Constraints (cont'd.)

- **`FOREIGN KEY`** clause
  - Default operation: reject update on violation
  - Attach **referential triggered action** clause
    - Options include `SET NULL`, `CASCADE`, and `SET DEFAULT`
    - Action taken by the DBMS for `SET NULL` or `SET DEFAULT` is the same for both `ON DELETE` and `ON UPDATE`
    - `CASCADE` option suitable for "relationship" relations

Specifying Constraints on Tuples

- NOT NULL Constraint – Ensures that a column cannot have NULL value.
- DEFAULT Constraint – Provides a default value for a column when none is specified.
- UNIQUE Constraint – Ensures that all values in a column are different.
- PRIMARY Key – Uniquely identifies each row/record in a database table.
- FOREIGN Key – Uniquely identifies a row/record in any of the given database table.

Specifying Constraints on Tuples

- `CHECK` clauses at the end of a `CREATE TABLE` statement
  - Apply to each tuple individually
  - `CHECK (Dept_create_date <= Mgr_start_date);`

INSERT, DELETE, and UPDATE Statements in SQL

- Three commands used to modify the database:
  - `INSERT`, `DELETE`, and `UPDATE`

The INSERT Command

- Specify the relation name and a list of values for the tuple

```
U1:   INSERT INTO   EMPLOYEE
      VALUES        ( 'Richard', 'K', 'Marini', '653298653', '1962-12-30', '98
                      Oak Forest, Katy, TX', 'M', 37000, '653298653', 4 );
```

```
U3B:  INSERT INTO   WORKS_ON_INFO ( Emp_name, Proj_name,
                    Hours_per_week )
      SELECT        E.Lname, P.Pname, W.Hours
      FROM          PROJECT P, WORKS_ON W, EMPLOYEE E
      WHERE         P.Pnumber=W.Pno AND W.Essn=E.Ssn;
```

The DELETE Command

- Removes tuples from a relation
  - Includes a `WHERE` clause to select the tuples to be deleted

```
U4A:  DELETE FROM   EMPLOYEE
      WHERE         Lname='Brown';

U4B:  DELETE FROM   EMPLOYEE
      WHERE         Ssn='123456789';

U4C:  DELETE FROM   EMPLOYEE
      WHERE         Dno=5;

U4D:  DELETE FROM   EMPLOYEE;
```

The UPDATE Command

- Modify attribute values of one or more selected tuples
- Additional **`SET`** clause in the `UPDATE` command
  - Specifies attributes to be modified and new values

```
U5:   UPDATE   PROJECT
      SET      Plocation = 'Bellaire', Dnum = 5
      WHERE    Pnumber=10;
```


Thank
You