DAMG 6210
Database Design

Nik Bear Brown

@NikBearBrown

Topics

- SQL

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

- The SELECT statement is used to retrieve data from the database.
- The basic syntax is:

```
SELECT <columnName>, <columnName>
FROM <TableName>

SELECT StudentFirstName, StudentLastName, StudentPhone
FROM Student
```

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.

Select All Columns, All Rows

List full details of all staff.

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

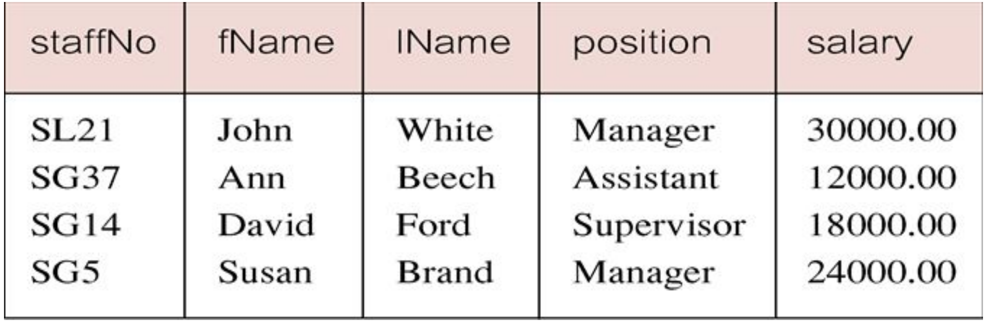
- Can use * as an abbreviation for ‘all columns’:

```
SELECT *
FROM Staff;
```

Select Comparison Search Condition

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

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

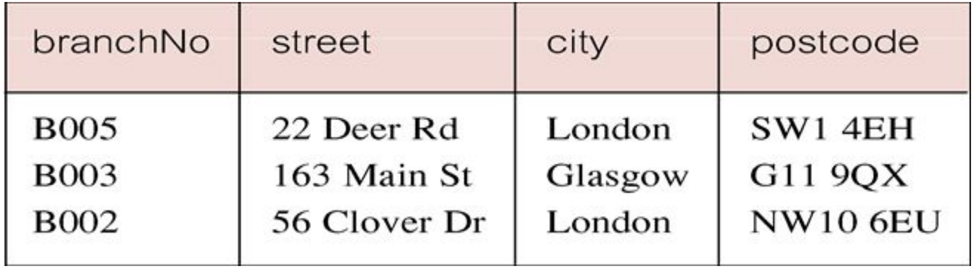


staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00
SG37	Ann	Beech	Assistant	12000.00
SG14	David	Ford	Supervisor	18000.00
SG5	Susan	Brand	Manager	24000.00

Select 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 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.

Select Range Search Condition

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

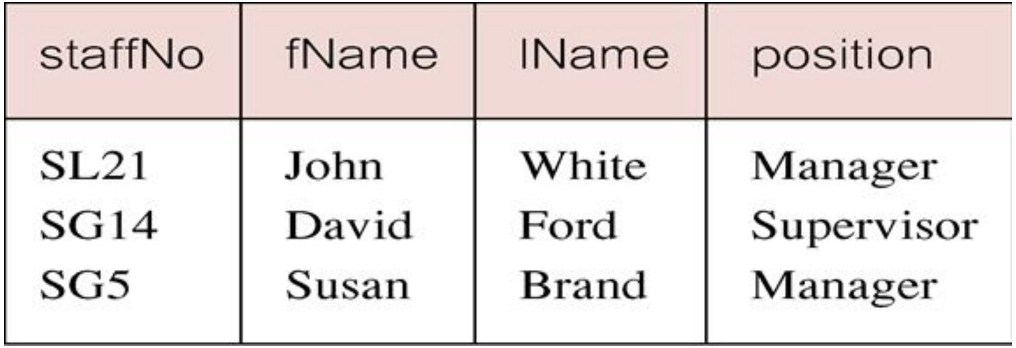
```
 SELECT staffNo, fName, lName, position, salary
FROM Staff
WHERE salary>=20000 AND salary <= 30000;
```

- Useful, though, for a range of values.

Select 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

The * WildCard

- Instead of listing each of the columns, you can use an * to include all columns.

```
SELECT * FROM
```

- Listing the columns does give you the ability to choose both which columns and which order to present them.
- With the * you return all the columns in the order they have in the underlying table.

Distinct Key Word

- Sometimes a query will return multiple duplicate values.
- For instance the statement

```
SELECT Key
FROM Session
```

  Could return numerous instances of each customer.

- The DISTINCT keyword will make it so it only returns one instance of each Key.

Distinct Key Word

```
SELECT DISTINCT Key
FROM Session
```

- The DISTINCT keyword always operates on the whole role, not on individual columns.
- It only returns distinct rows.

Calculations

- You can do calculations in SELECT statements.

```
SELECT ItemNumber, ItemPrice, Quantity, ItemPrice *
Quantity
FROM CustomerOrder
```

Order of Operations

The order of operation is the same as in algebra.

1. Whatever is in parentheses is executed first. If parentheses are nested, the innermost is executed first, then the next most inner, etc.
2. Then all division and multiplication left to right
3. And finally all addition and subtraction left to right

Sorting

- You can sort the results of a query by using the keywords ORDER BY.

```
SELECT *
FROM Session
ORDER BY SessionDate
```

- ORDER BY does an ascending A-Z, 1-10, etc. sort by default.
- You can change the direction by using the DESC keyword after the field to be sorted.

Aliasing

- Sometimes it is useful to alias a column name to make a more readable result set.

```
SELECT StudentLastName AS [Last Name], StudentFirstName AS
[First Name]
FROM Student
```

- The AS keyword is optional.
- Double quotes “ “ can be used instead of square brackets.

Where Clause

- The WHERE clause allows you to limit the rows you return in a query.
- You use the WHERE clause to specify the criteria by which the rows will be filtered.

```
SELECT LastName, FirstName, Phone, City
FROM Customer
WHERE City = ‘Boston’
```

Other Criteria

- As well as equal you can use other operators for the criteria:

  >

  <

  >=

  =<

- Character and date values in the criteria are quoted with single quotes.
- Numerical values are not quoted.

Like

- The LIKE keyword used in a WHERE operator with a wildcard (% or _) allows you to search for patterns in character-based fields.
- The following returns all items whose name starts with “T.”

```
SELECT ItemName, ItemPrice
FROM Inventory
WHERE ItemName LIKE ‘T%’
```

Between

- The BETWEEN keyword can be used in criteria to return values between to other values.
- BETWEEN is inclusive of its ends.

```
SELECT Key, SessionDate, StudentKey
FROM Session
WHERE SessionDate BETWEEN ’11/1/2014’ AND
‘11/30/2014’
```

AND OR NOT

- You can use keywords AND, OR, and NOT to combine criteria in a query.
- AND is exclusive. Or is Inclusive.
- WHERE City = ‘Boston’ OR City=‘Los Angeles’ returns all records that have either Boston or Los Angeles for their city.
- WHERE City=‘Boston’ AND City=‘Los Angeles’ returns nothing because the record cannot have both at the same time.
- NOT excludes.
- WHERE NOT City = ‘Los Angeles’ returns every city except Los Angeles.

NULL

- Nulls are special cases. They are not a value and so cannot be compared to a value using = or < or >.
- To locate nulls you can use the IS keyword in a criteria:

```
WHERE StudentKey IS NULL
WHERE StudentKey IS NOT NULL
```

Select Set Membership

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

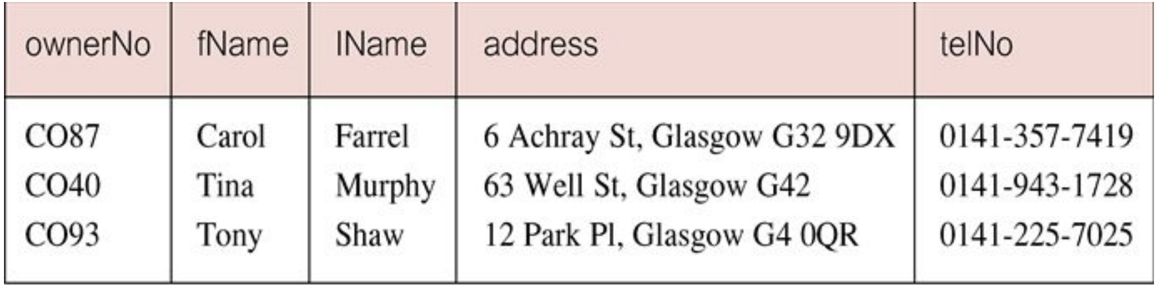
```
SELECT staffNo, fName, lName, position
FROM Staff
WHERE position=‘Manager’ OR
            position=‘Supervisor’;
```

- IN is more efficient when set contains many values.

Select 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

Select Pattern Matching

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

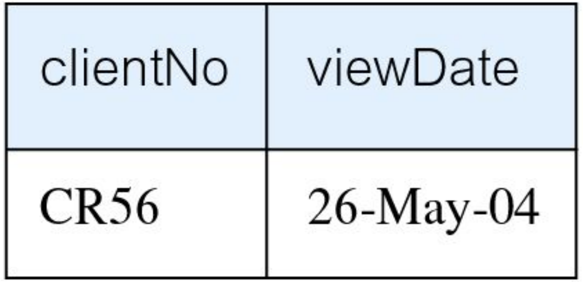
Select NULL Search Condition

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;
```

Select NULL Search Condition



clientNo	viewDate
CR56	26-May-04

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

Select 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;
```

Select Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

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

Select Multiple Column Ordering

To arrange in order of rent, specify minor order:

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

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.

SELECT Statement - Aggregates

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

SELECT Statement - Aggregates

- Aggregate functions can be used only in SELECT list and in HAVING clause.

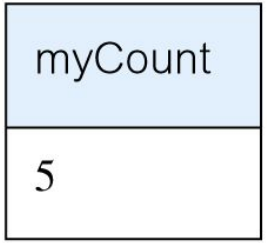
- If SELECT list includes an aggregate function and there is no GROUP BY clause, SELECT list cannot reference a column out with an aggregate function. For example, the following is illegal:

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

Select - Use of COUNT(*)

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

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

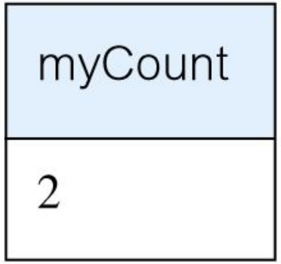


myCount
5

Select - Use of COUNT(DISTINCT)

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’;
```

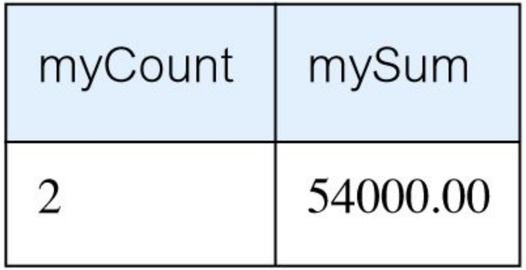


myCount
2

Select - 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’;
```

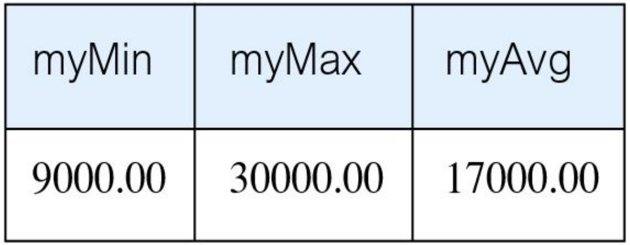


myCount	mySum
2	54000.00

Select - 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

SELECT Statement - Grouping

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

SELECT Statement - Grouping

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

Select - 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;
```

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.

Select - 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;
```

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.

Select - 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’);
```

Select - 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’;
```

Select - 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);
```

Select - 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;
```

Subquery Rules

- ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).

- Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.

- By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.

Subquery Rules

- When subquery is an operand in a comparison, subquery must appear on right-hand side.

- A subquery may not be used as an operand in an expression.

Select - 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 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.

Select - 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’);
```

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’);
```

Functions

- Functions always have the same basic syntax:

  <function name>(function arguments)
- There are hundreds of built-in functions.
- We will be concerned with two broad types of functions:
  - Scalar functions
  - Aggregate functions

Scalar Functions

- Scalar functions operate on a single row at a time.
- Here is a list of scalar functions used in this chapter.

Function Name	Description
GETDATE()	Returns current date and time
MONTH	Returns the month as an integer (1 to 12) from a Date value
YEAR	Returns the Year as a four-digit integer from a date value

Aggregate Functions

- Aggregate functions operate on multiple rows at a time.
- Here is a table of common aggregate functions:

Aggregate Function	Description
COUNT	Counts the number of values : COUNT(*) counts all the rows. COUNT(columnName) counts all the values in the column but ignores nulls
SUM	Sums or totals numeric values: SUM (InStock)
AVG	Returns the mean average of a set of numeric values: AVG(Price). By default nulls are ignored.
MAX	Returns the highest value in a set of numeric or datetime values: MAX(price)
MIN	Returns the smallest value in a set of numeric or datetime values: MIN(Price)

Using Distinct in Aggregate Functions

- You can use the DISTINCT keyword with aggregate functions.
- Doing so means the function will ignore duplicate values in its calculation.

```
SELECT COUNT(DISTINCT StudentKey) AS [Unduplicated]
FROM Session
```

Group By

- When a SELECT clause includes an aggregate function and columns that are not a part of that function, you must use the GROUP BY keywords to group by each of the non-included columns.
- This is necessary because you are mixing functions that operate on multiple rows with columns that refer to values in individual rows only.

Group By Example

```sql
SELECT Key, COUNT(SessionTimeKey) AS [Total Sessions]

FROM Session

GROUP BY Key
```

Having

- The HAVING keyword is used when there is an aggregate function in the criteria of a query.

```
SELECT Key, COUNT(SessionTimeKey) AS [Total Sessions]
FROM Session
GROUP BY Key
HAVING COUNT(SessionTimeKey)<4
```

EXISTS and NOT EXISTS

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

EXISTS and NOT EXISTS

- As (NOT) EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.

- Common for subqueries following (NOT) EXISTS to be of form:

  (SELECT * ...)

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’);
```

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;
```

Query using EXISTS

- Could also write this query using join construct:

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

Joins

- In database design and normalization, the data are broken into several discrete tables.
- Joins are the mechanism for recombining the data into one result set.
- We will look at three kinds of joins:
  - Inner joins
  - Equi joins
  - Outer joins

Multi-Table Queries

- Can use subqueries provided result columns come from same table.

- If result columns come from more than one table must use a join.

- To perform join, include more than one table in FROM clause.

- Use comma as separator and typically include WHERE clause to specify join column(s).

Multi-Table Queries

- Also possible to use an alias for a table named in FROM clause.

- Alias is separated from table name with a space.

- Alias can be used to qualify column names when there is ambiguity.

Simple Join

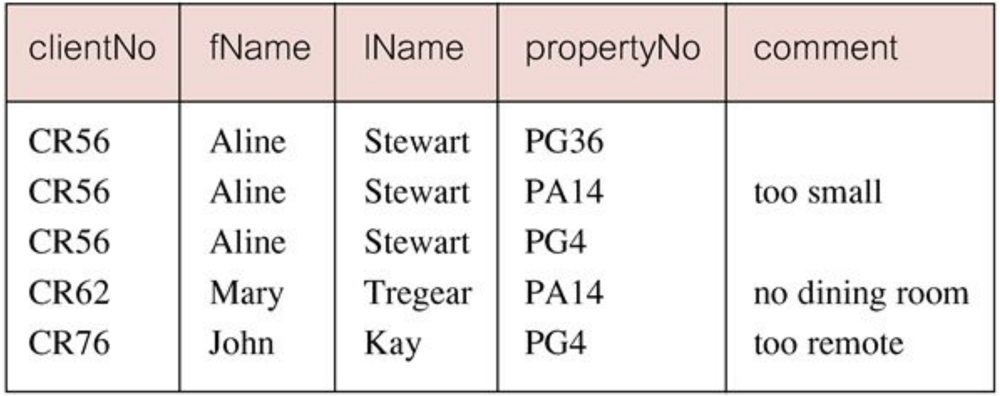
List names of all clients who have viewed a property along with any comment supplied.

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

Simple Join

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

- Equivalent to equi-join in relational algebra.



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

Basic INNER JOIN Syntax

```
SELECT <column1, column2>
FROM <table1>
INNER JOIN <table2>
ON <table1>.<column>=<table2>.<column>
```

Inner Joins

- Inner joins return related records from each of the tables joined.

```
SELECT LastName,
FirstName,
SessionDateKey,
SessionTimeKey,
StudentKey
SessionStatus
FROM
INNER JOIN Session
ON .Key = Session.Key
```

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.

Equi Joins

- Equi joins present an alternative way to perform inner joins. Some older RDMSs only support this alternative form. The example below also uses an alias for the table name.

```
SELECT t.Key,
LastName,
FirstName,
SessionDateKey,
SessionTimeKey,
StudentKey
FROM  t,
Session s
WHERE t.Key = s.Key
AND LastName = ‘Brown’
```

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;
```

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;
```

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;
```

Computing a Join

Procedure for generating results of a join are:

1. Form Cartesian product of the tables named in FROM clause.

2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.

3. For each remaining row, determine value of each item in SELECT list to produce a single row in result table.

Computing a Join

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

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

- SQL provides special format of SELECT for Cartesian product:

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

Outer Joins

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

Branch1

branchNo	bCity
B003	Glasgow
B004	Bristol
B002	London

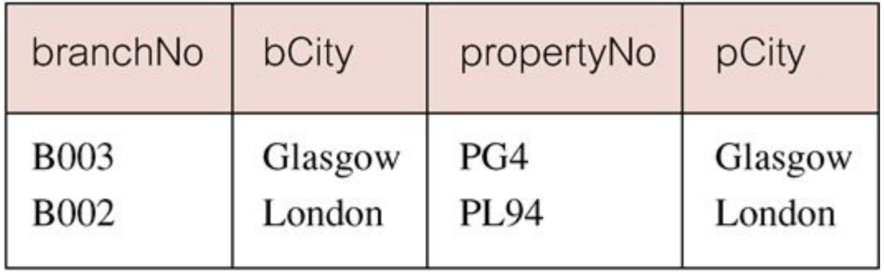
PropertyForRent1

propertyNo	pCity
PA14	Aberdeen
PL94	London
PG4	Glasgow

Outer Joins

- The (inner) join of these two tables:

```
SELECT 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.

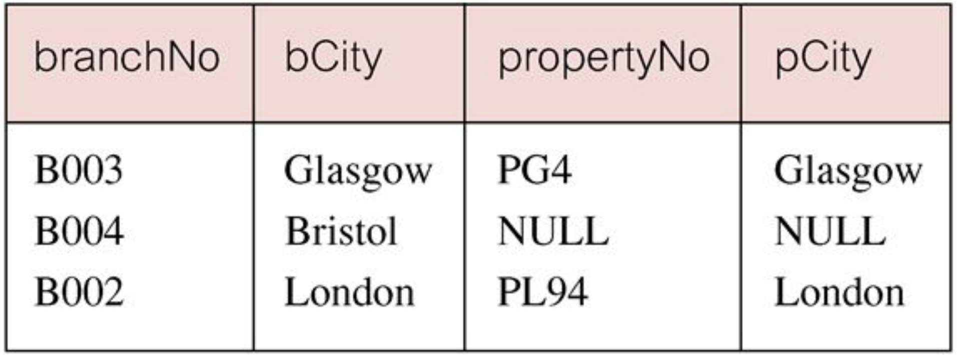
Left Outer Join

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

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

Left Outer Join

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



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

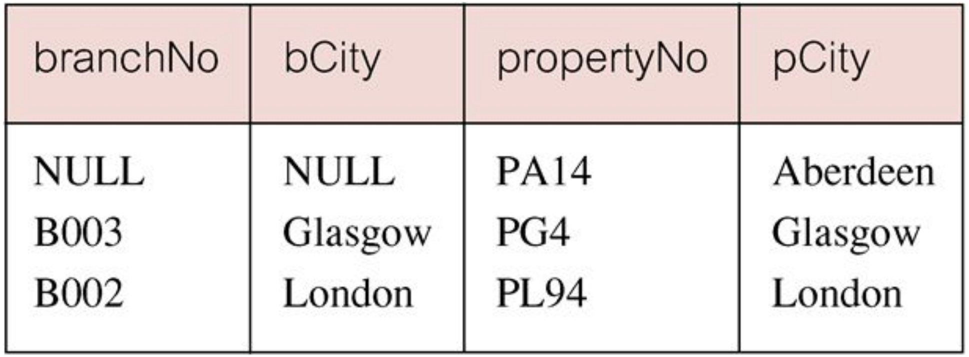
Right Outer Join

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

- 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

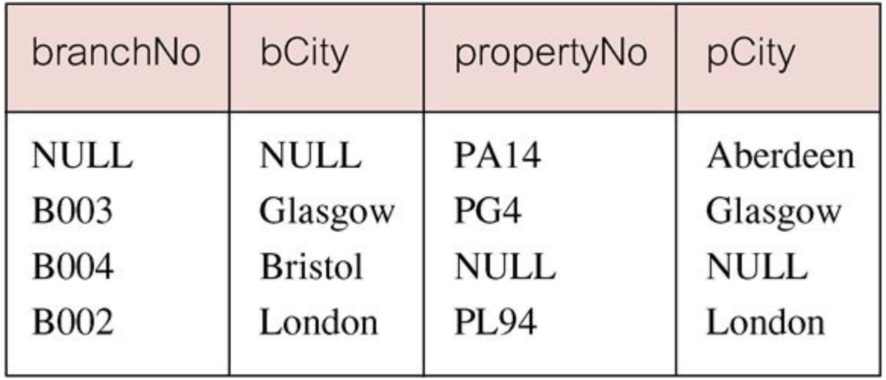
Full Outer Join

List branches and properties in same city and any unmatched branches or properties. (on both sides)

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

Full Outer Join

- 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

OUTER JOIN Syntax

Outer joins return records that are not matched. The following query returns s that have no sessions scheduled.

```
SELECT <column1>, <column2>
FROM <table1>
LEFT OUTER JOIN <table2>
ON <table1>.<column>=<table2>.<column>
```

Outer Join Example

```
SELECT t.Key,
LastName,
SessionDateKey
FROM  t
LEFT OUTER JOIN Session s
ON t.Key = s.Key
WHERE SessionDateKey IS Null
```

Inserts

To insert a record into a table, you use the following syntax:

```
INSERT INTO <tablename>(<ColumnName>,
<columnName>, ...)
VALUES(<value1>, <value2>, ...)
```

Updates

Updates allow you to change existing records. The syntax is:

```
UPDATE <TableName>
SET <ColumnName> = <New Value>,
<ColumnName>=<new value>
WHERE <ColumnName> = <criteria>
```

Deletes

- Deletes allow you to remove a record from a table:

```
DELETE FROM <TableName>
WHERE <columnName> = <criteria>
```

Deletes and Updates

- Deletes and updates are dangerous. If you do not specify a criteria, the update or delete will be applied to all the rows in a table.
- Also, referential integrity may prevent a deletion. You cannot delete a parent that has children in another table.

SubQuery Example

```
SELECT DISTINCT COUNT(*) AS Total,
(SELECT COUNT(*)
FROM Session
WHERE SessionStatus='NS') AS NoShow,
(SELECT COUNT(*)
FROM Session
WHERE SessionStatus='c') AS Completed
FROM Session
```

This example shows subqueries used in the SELECT clause to return Aggregate values.

Locating Duplicates

```
SELECT Lastname, firstname, email, phone,
COUNT(*) AS [duplicates]
FROM contact
GROUP BY Lastname, firstName, email, Phone
HAVING COUNT(*) >1
```

This SQL finds duplicate values in in a table.

Documentation: Testing Plans

- When testing the database, you should document all your SQL queries and their results.
- On the next slide is a sample of a test table, showing the test and results.

Union, Intersect, and Difference

- 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*.

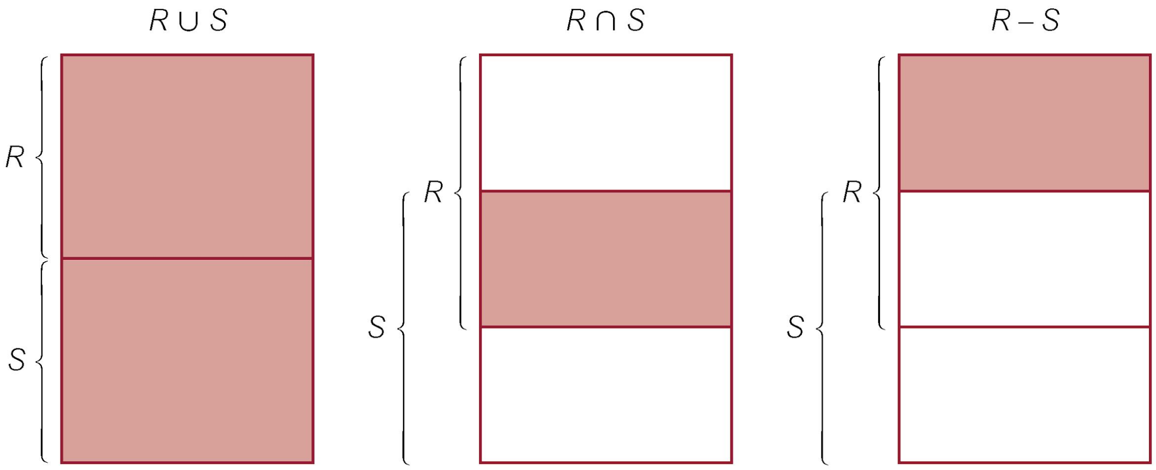
Union, Intersect, and Difference

- 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



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

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);
```

Use of UNION

- Or

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

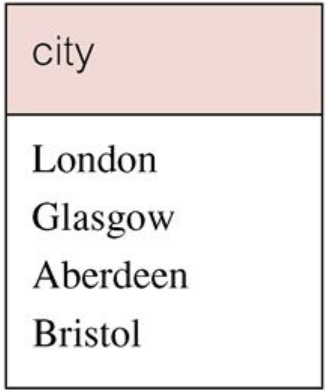
Union Example

```
INSERT INTO Contact(LastName, FirstName, Email, Phone)
SELECT StudentLastName AS LastName,
StudentFirstName AS FirstName,
StudentEmail AS Email,
StudentPhone AS Phone
FROM Student
WHERE StudentEmail IS NOT NULL
UNION
SELECT LastName,
FirstName,
Email,
Phone
FROM
WHERE Email IS NOT NULL
```

This UNION query joins the tables Student and into a single result and writes them to the table Contact.

Use of UNION

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



city
London
Glasgow
Aberdeen
Bristol

Use of INTERSECT

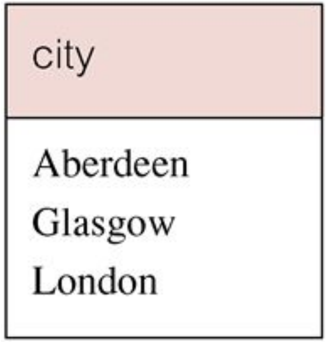
List all cities where there is both a branch office and a property.

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

Use of INTERSECT

- Or

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



city
Aberdeen
Glasgow
London

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);
```

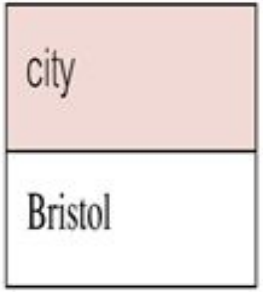
Use of EXCEPT

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

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.

INSERT

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

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’);
```

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 ...
```

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.

INSERT … SELECT

```
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));
```

UPDATE

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

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

UPDATE

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

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’;
```

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.

DELETE Specific Rows

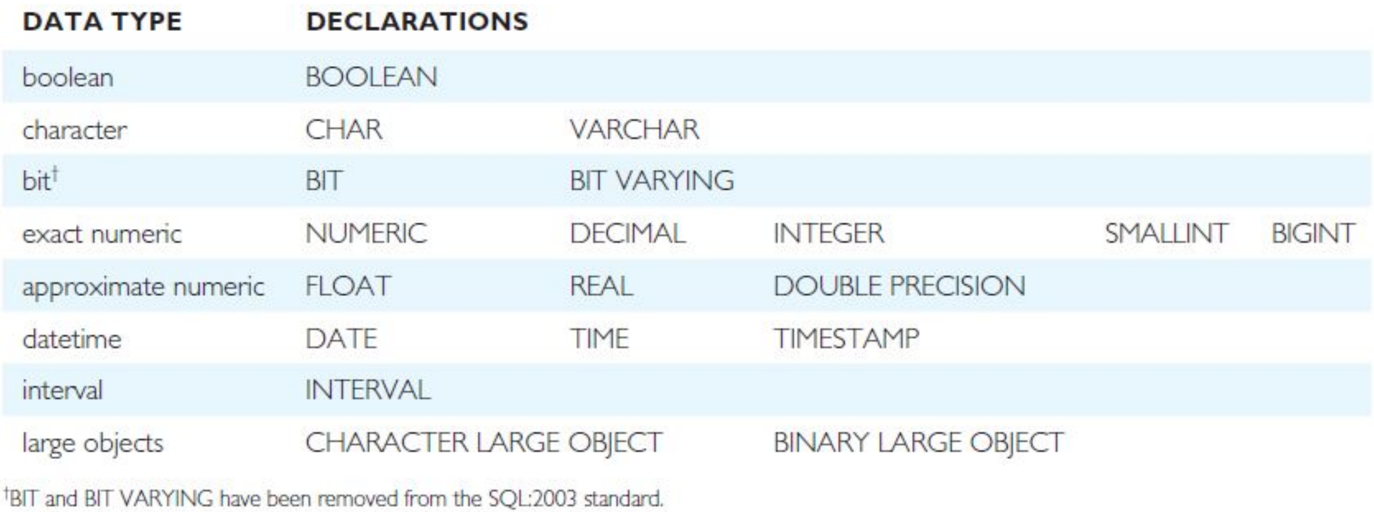
Delete all viewings that relate to property PG4.

```
DELETE FROM Viewing
 WHERE propertyNo = ‘PG4’;
```

Delete all records from the Viewing table.

```
DELETE FROM Viewing;
```

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.

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.

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.

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’;
```

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

<u>View</u>

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 <u>defining query</u>.
- 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.