SQL

Learning Outcomes

- Understand the purpose and importance of SQL.
- Be able to retrieve data from database and formulate queries using SELECT and:
  - Use compound WHERE conditions.
  - Sort query results using ORDER BY.
  - Use aggregate functions.
  - Group data using GROUP BY and HAVING.
  - Join tables together.
  - Use subqueries.
- Be able to update database and formulate queries using INSERT, UPDATE, and DELETE.

Introduction to SQL

- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining database structure.
  - A DML for retrieving and updating data.
- 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.

Introduction to 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;
```

- 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.

Introduction to SQL

- Data definition language (DDL)
  - Create table
  - Drop table
- Data manipulation language (DML)
  - Insert
  - Delete
  - update
  - Select

Basic SELECT Statement

```
SELECT A1, A2, …., An
FROM R1, R2, …., Rn
WHERE condition
```

Eg 1. All Columns, All Rows

- List full details of all staff.

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

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

```
SELECT * FROM Staff;
```

- Results:

```
mysql> select * from staff;
+---------+-------+-------+------------+------+------------+--------+-----------------+
| staffNo | fName | lName | position   | sex  | DoB        | salary | BRANCH_BranchNo |
+---------+-------+-------+------------+------+------------+--------+-----------------+
| SA9     | Mary  | Howe  | Assistant  | F    | 1970-02-19 |   9000 | B007            |
| SG14    | David | Ford  | Supervisor | M    | 1958-03-24 |  18000 | B003            |
| SG37    | Ann   | Beech | Assistant  | F    | 1960-11-10 |  12000 | B003            |
| SG5     | Susan | Brand | Manager    | F    | 1940-06-03 |  24000 | B003            |
| SL21    | John  | White | Manager    | M    | 1945-10-01 |  30000 | B005            |
| SL41    | Julie | Lee   | Assistant  | F    | 1965-06-13 |   9000 | B005            |
+---------+-------+-------+------------+------+------------+--------+-----------------+
6 rows in set (0.00 sec)
```

Eg 2. Specific Columns, All Rows

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

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

- Results:

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

Eg 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

Eg 3. Use of DISTINCT

- Use DISTINCT to eliminate duplicates:

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

propertyNo
PA14
PG4
PG36

Eg. 4 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;
```

Eg. 5 Comparison Search Condition

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

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

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

Eg. 6 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

Eg. 7 Range Search Condition

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

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

- BETWEEN test includes the endpoints of range.

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

Eg. 8 Pattern Matching

Find all owners whose address is in Glasgow. (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

Eg. 8 Pattern Matching

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

Eg. 9 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;
```

Eg. 9 NULL Search Condition

clientNo	viewDate
CR56	26-May-04

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

Eg. 10 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

Eg. 11 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

Eg. 11 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;
```

Eg. 11 Multiple Column Ordering

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.

Eg. 12 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

Eg. 13 Use of COUNT(DISTINCT)

How many different properties viewed in May ‘04?

```
SELECT
COUNT(DISTINCT propertyNo) AS myCount
FROM Viewing
WHERE viewDate BETWEEN “2004-05-01”
                   AND “2004-05-31”;
```

myCount
2

Eg. 14 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

Eg. 15 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.

Eg. 16 Use of GROUP BY

Find total 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;
```

Eg. 16 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 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.

Eg. 17 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;
```

Eg. 17 Use of HAVING

branchNo	myCount	mySum
B003	3	54000.00
B005	2	39000.00

Multi-Table Queries

- 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.

Eg. 18 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;
```

Eg. 18 Simple Join

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

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

Eg. 19 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;
```

Eg. 19 Sorting a join

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

Eg. 20 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;
```

Eg. 20 Three Table Join

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

Eg. 21 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;
```

Eg. 21 Multiple Grouping Columns

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:

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.
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.

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.

Eg. 22 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’);
```

Eg. 22 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’;
```

Eg. 22 Subquery with Equality

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

Eg. 23 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);
```

Eg. 23 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;
```

Eg. 23 Subquery with Aggregate

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.

Eg. 24 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”));
```

Eg. 24 Nested subquery: use of IN

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.

Eg. 25 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');
```

Eg. 25 Use of ANY/SOME

- 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

Eg. 26 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');
```

staffNo	fName	lName	position	salary
SL21	John	White	Manager	30000.00

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 * ...)

Eg. 27 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');
```

Eg. 27 Query using EXISTS

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

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;
```

Eg. 27 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';
```

Summary of SELECT statement

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

DML - UPDATE

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

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

UPDATE

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

Example: 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';
```

DML - 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;
```

DML – 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.

Example: INSERT … VALUES

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

```
INSERT INTO BRANCH
VALUES ('B005', '22 Deer Rd', 'London', 'SW1 4EH');
```

Note:

- The DATE type in MySQL follows the 'YYYY-MM-DD' format. The supported range is '1000-01-01' to '9999-12-31'.

```
INSERT INTO Staff
VALUES ('SL21', 'John', 'White', 'Manager', 'M',
'1945-10-01', 30000, 'B005');
```

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');
```

SQL Data definition (DDL)

- **Objects**
  - Table
- **Commands**
  - CREATE
  - ALTER
  - DROP

DDL – Create Table

- Created by CREATE TABLE statement

  CREATE TABLE Branch;

- Attributes ordered by creation order
- Rows not ordered

DDL – Create Table

CREATE TABLE *table_name*
({ *column_name data_type*
[DEFAULT *default_expr*] [
column_constraint [, ...]] }
|table_constraint } [, ...])

Example:

```
CREATE TABLE `branch` (
  `BranchNo` char(4) NOT NULL,
  `street` varchar(45) DEFAULT NULL,
  `city` varchar(45) DEFAULT NULL,
  `postcode` varchar(45) DEFAULT NULL,
  PRIMARY KEY (`BranchNo`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
```

DDL – Drop Table

Syntax:

DROP [TEMPORARY] TABLE [IF EXISTS]

tbl_name [, tbl_name] ...

[RESTRICT | CASCADE]

What have we learned?

- SQL DML:
  - SELECT statement
  - Key words: DISTINCT, calculated field, pattern matching (%, _), search for NULL, ORDER BY, GROUP BY, HAVING
  - Aggregate functions: COUNT, MIN, MAX, AVG, SUM
  - Subqueries: IN, ANY (SOME), ALL, EXISTS
  - UPDATE
  - DELETE
  - INSERT
- SQL DDL: CREATE table, DROP table