SQL - Structured Query Language

It was designed by IBM. It is:

- the standard relational language
- declarative (i.e. you just describe the desired result)
- not just a query language,
  - but also for data definition and manipulation

It has many dialects - but now has standards:

- SQL89 (SQL1) - first attempt at a general standard
- SQL92 (SQL2) – the one most DBMS say they implement
  - but Oracle (e.g.) has slight differences, notably in domain names
- SQL99 (SQL3) - extension to a full object oriented programming language

Each DBMS conforms to a subset of the standards, and can introduce additional features

Oracle SQL+ : The Main Commands

1. Data Definition Commands *Lecture 17*

Create Commands - data definition commands which create a database object

- e.g. create table - creates a new table

Alter Commands - data definition commands which allow database objects to be edited

Drop Commands - data definition commands which delete database objects

2. Data Manipulation Commands *Lecture 17*

Insert - adds a new record to a table

Delete - removes records from a table

Update - changes the values in records in a table

Truncate - remove all rows from a table (faster than Delete *)

3. Query Commands *Lectures 15 and 16*

Select - retrieves data from the database

4. Transaction Commands *a later topic*

Commit – ends current transaction, making all changes permanent

Rollback - undo all work in this transaction

Savepoint - create a point within a transaction to which you can roll back

5. Security Commands *a later topic*

Grant - give a user or role access to a table or view

Revoke - remove granted access

Overview of Syntax

SQL commands are **not case sensitive**, but a recommended approach is to have

- SQL words in uppercase
- Table names starting with a capital letter (like a java class)
- Column names starting with lowercase (like a java variable)
- but note your **data** will be case sensitive
  - e.g. if you want to test a gender column for males and you ask
    WHERE gender = 'm'
    then you'd better have lower case letters in your database or put
    OR gender = 'M'

Syntax:

- All SQL Commands are terminated by a semi-colon
  (except in AquaData which uses the GO command)
- Strings are in single quotes

Lots of examples in the text file CreateAllTables.sql

Querying In SQL

The basic form of query is :

SELECT ***List of expressions*** *These are usually just column names, then this is equivalent to projection*

FROM ***List of tables*** *This forms a Cartesian Product*

WHERE ***condition*** *This makes a selection*

e.g.

```
SELECT dateofBirth, house, street
        FROM  Employee, Department
                WHERE city = 'Glasgow' AND dept = deptID
                        AND deptname = 'Admin';
```

The order of interpretation is:

- (FROM) I want to look at all combinations of records in these tables
- (WHERE) Filter out some of them using the condition – *like Java* ***if***
- (SELECT) Then show me the data I have asked for

More on Querying

The **WHERE** clause can be dropped if there is no selection.

Use **AND, OR, brackets** etc for complex conditions

We can retrieve all the columns by use of *****

```
SELECT *  FROM Employee
    WHERE city = 'Glasgow'
        AND gender = 'F';
```

To return everything from a table, you would do, e.g.:

```
SELECT *  FROM Employee;
```


Duplicates in Retrievals

In general, a query returns the result **including** duplicated records

- i.e. it isn't really a relation – this must be a **set** of records
- instead this is (mathematically) a **bag** of records
- which is vital if you want to return everything

To eliminate duplicates include the word **DISTINCT**, e.g.

```
SELECT gender FROM Employee
```

returns many rows containing 'M' or 'F'

```
SELECT DISTINCT gender FROM Employee
```

returns at most the two rows : 'M' and 'F'

Using Explicit Sets

We can write explicit sets instead of sub-queries

```
SELECT name FROM Employee
    WHERE dept IN (1,2,3);
```

Nulls in Queries

We can do:

```
SELECT name FROM Employee
    WHERE supervisor IS NULL;
```

i.e. Give the names of employees without supervisors, or alternatively:

```
WHERE supervisor IS NOT  NULL;
```

See slides 373-376 for anomalies in queries involving nulls

Wildcard Characters and Pattern Matching

```
SELECT ni# FROM Employee
    WHERE (lName LIKE 'A%') AND (fName LIKE '_on')
```

picks the employees whose last names begin with A and whose first names have three letters, the last two of which are "on" - e.g. "Ron" and "Don".

i.e. "%" is a wildcard character which matches **any number of characters**, while "_" is a wildcard character which matches **a single character**

- NB Other DBMS may use different wildcard characters

Using Range Tests

We can limit the values of a column to lie within a specific range:

```
SELECT lName FROM Employee
   WHERE dept BETWEEN 5 AND 7        //inclusive
```

- which picks the employees whose work for departments 5, 6 and 7

NOT can be used with any of these - e.g. NOT IN, NOT BETWEEN, NOT LIKE, etc.

Functions in Oracle SQL

conversion functions - e.g. to_char (convert anything to a textual representation)

numeric functions - abs, ceil, cos, cosh, exp, floor, ln, log, mod, power, round, sign, sin, sinh, sqrt, tan, tanh, trunc

character functions - e.g. substr, length, lower, lpad

date functions - to extract week, month, quarter or year, day, time…

aggregate functions - count, sum, avg, max, min, glb, lub, stddev, variance

other functions - e.g. user (return current user)

Examples of Using Oracle Functions

Here are some that may be useful.

Compare text ignoring case:

```
SELECT * FROM Employee
    WHERE UPPER(city) = 'GLASGOW';
```

- This converts the data to upper case before comparing it, so (provided that the literal value is also completely in upper case) the comparison is not case sensitive
- The function **LOWER** can be used similarly

Rounding calculation results

```
SELECT ROUND(AVG(balance),1)
```

- second parameter gives number of decimal places
- use 0 to get an integer

Formatting Output

It is often sensible to rename columns to give better headings, e.g.

```
SELECT bDate AS birthdate
    FROM Employee;
```

You can use the **SUBSTR** function to truncate a field

```
SELECT SUBSTR(fName,1,1) AS initial, lName as 'last name'
    FROM Employee;                    NB renaming
```

Alternatively you can use the SQL*Plus Format command:

```
COLUMN fName FORMAT A1      (char length = 1)

SELECT fName, lName  FROM Employee;
```

For landscape output use the SQL*Plus command

```
SET LINESIZE 120
```

The Semantics of NULL Comparisons

Remember tuples in SQL relations can have NULL as a value for one or more components

- as long as the variable hasn't been declared PRIMARY KEY or NON NULL

The meaning depends on context. Two common cases:

- *Missing value*: We don't know all the values in the column:
  - e.g., we don't know always know the address of each employee
- *Inapplicable*: Not all tuples have a value for this column
  - e.g., the value of attribute spouse for an unmarried person.

When comparing NULLs to values we will have to use a three-valued logic: **`TRUE, FALSE, UNKNOWN`**

- When any value is compared with `NULL`, the truth value is `UNKNOWN`
- But a query only produces a tuple in the answer if its truth value for the `WHERE`-clause evaluates to **`TRUE`** (not `FALSE` or `UNKNOWN`).

Examples of Using Nulls

Assuming that the Employee table allows null values for all fields except the employee number and the first and last name, then

- SELECT fname, lname FROM Employee WHERE age > 0
  - will only return the names of employees whose ages are not NULLs
- SELECT dept FROM Employee
  WHERE fname = 'John' AND age > 30
  - will also omit any employee whose age we don't know
- SELECT dept FROM Employee
  WHERE fname = 'John' OR age > 30
  - will include all Johns

Truth Tables

AND	TRUE	FALSE	UNK
TRUE	TRUE	FALSE	UNK
FALSE	FALSE	FALSE	FALSE
UNK	UNK	FALSE	UNK

OR	TRUE	FALSE	UNK
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNK
UNK	TRUE	UNK	UNK

Three-Valued Logic

To understand how AND, OR, and NOT work in 3-valued logic, think of:

TRUE = 1, FALSE = 0, and UNKNOWN = ½

and **AND = MIN; OR = MAX; NOT(*x*) = 1-*x***

Example:

TRUE AND (FALSE OR NOT(UNKNOWN)) =
MIN(1, MAX(0, (1 - ½))) =
MIN(1, MAX(0, ½) = MIN(1, ½) = ½

Explanation:

- NOT(UNKNOWN) evaluates to UNKNOWN because we still don't know whether the value is TRUE or FALSE
- FALSE or UKNOWN is also unknown because it still may be TRUE or FALSE
- TRUE and UNKNOWN is also unknown because we if don't the second atom is TRUE or FALSE and so we don't know the result of ANDing with TRUE

Anomalies of Three Valued Logic

2-valued Laws != 3-Valued Laws

- Some common laws, like the commutativity of AND (i.e. X AND Y = Y AND X), hold in 3-valued logic
  - so **TRUE AND UNKNOWN = UNKNOWN AND TRUE**
- But others do not; example: the "law of excluded middle",

  ***p* OR NOT *p* = TRUE**
- For 3-valued logic, the result might be UKNOWN:
  - if *p* = `UNKNOWN`, then left side = **max(1/2,(1–1/2)) = 1/2 != 1**

Example

- **SELECT name FROM Person WHERE age<18 OR age >= 18**
  - will omit any people who age column is NULL
  - because we are testing UNKNOWN OR UNKNOWN

Like bag algebra, there is no way known to make 3-valued logic conform to all the laws we expect for sets/2-valued logic, respectively