STRUCTURED QUERY LANGUAGE (SQL)

Kapukha

Database Language

- A database language should allow a user to:
  - Create the database and relation structures;
  - Perform basic data management tasks, such as the insertion, modification, and deletion of data from the relations;
  - Perform both simple and complex queries.
- A database language must perform these tasks with minimal user effort, and its command structure and syntax must be relatively easy to learn.
- The language must be portable

SQL

- SQL is an example of a transform-oriented language, or a language designed to use relations to transform inputs into required outputs.
- It is a non-procedural language: you specify what information you require, rather than how to get it.
- Like most modern languages, SQL is essentially free-format, which means that parts of statements do not have to be typed at particular locations on the screen.
- The command structure consists of Standard English words such as CREATE TABLE, INSERT, SELECT. F

SQL

- As a language, the ISO SQL standard has two major components:
    - A Data Definition Language (DDL) for defining the database structure and controlling access to the data;
    - A Data Manipulation Language (DML) for retrieving and updating data.
- The command structure consists of Standard English words such as CREATE TABLE, INSERT, SELECT.
- An SQL statement consists of reserved words and user-defined words.
    - Reserved words are a fixed part of the SQL language and have a fixed meaning. They must be spelt exactly as required and cannot be split across lines.
    - User-defined words are made up by the user (according to certain syntax rules) and represent the names of various database objects such as tables, columns, views, indexes,

DML

- DATA MANIPULATION This section looks at the SQL DML statements, namely:
  - SELECT – to query data in the database;
  - INSERT – to insert data into a table;
  - UPDATE – to update data in a table;
  - DELETE – to delete data from a table.
- Simple Queries: The purpose of the SELECT statement is to retrieve and display data from one or more database tables.
  - ***Staff (staffNo, fName, lName, position, sex, DOB, salary, branchNo)***

USING SELECT

- SELECT is the most frequently used SQL command and has the following general form:
  - ~~**SELECT [DISTINCT | ALL] {* | [columnExpression [AS newName]] [,**~~ **. . .]}**
  - **FROM TableName [alias] [, . . .]**
  - **[WHERE condition]**
  - **[GROUP BY columnList]**
  - **[HAVING condition] [**
  - **ORDER BY columnList]**
- Note: **columnExpression** represents a column name or an expression,
- **TableName** is the name of an existing database table or view that you have access to, and alias is an optional abbreviation for TableName.

SEQUENCE OF SELECT

- The sequence of processing in a SELECT statement is:
  - FROM specifies the table or tables to be used
  - WHERE filters the rows subject to some condition
  - GROUP BY forms groups of rows with the same column value
  - HAVING filters the groups subject to some condition
- SELECT specifies which columns are to appear in the output
- ORDER BY specifies the order of the output
- The order of the clauses in the SELECT statement cannot be changed.
- The only two mandatory clauses are the first two: SELECT and FROM; the remainder are optional.
- The SELECT operation is closed: the result of a query on a table is another table

Retrieve all rows

- Example 1: ***Retrieve all columns, all rows : List full details of all staff.***
- Since there are no restrictions specified in this query, the WHERE clause is unnecessary and all columns are required.
- We write this query as:
  - **SELECT staffNo, fName, lName, position, sex, DOB, salary, branchNo FROM Staff;**
- Since many SQL retrievals require all columns of a table, there is a quick way of expressing 'all columns' in SQL, using an asterisk (*) in place of the column names.
- The following statement is an equivalent and shorter way of expressing this query:
  - **SELECT * FROM Staff;**

Retrieve specific columns, all rows

- ***Example 2: Produce a list of salaries for all staff, showing only the staff number, the first and last names, and the salary details.***
  - SELECT staffNo, fName, lName, salary FROM Staff;

Use of DISTINCT

- ***Example 3: List the property numbers of all properties that have been viewed.***
  - SELECT propertyNo FROM Viewing;
- The result table will have are several duplicates because. To eliminate the duplicates, we use the **DISTINCT** keyword.
- Rewriting the query as:
  - SELECT DISTINCT propertyNo FROM Viewing;

Calculated fields

- ***Example 4: Produce a list of monthly salaries for all staff, showing the staff number, the first and last names, and the salary details.***
    - **SELECT staffNo, fName, lName, salary/12 FROM Staff;**
- In this case, the desired result can be obtained by simply dividing the salary by 12.
- In general, to use a calculated field you specify an SQL expression in the SELECT list.
- An SQL expression can involve **addition, subtraction, multiplication, and division**, and **parentheses** can be used to build complex expressions.
- More than one table column can be used in a calculated column; however, the columns referenced in an arithmetic expression must have a **numeric type**.
- In the previous example, we could have written:
    - **SELECT staffNo, fName, lName, salary/12 AS monthlySalary FROM Staff;**
- In this case the column heading of the result table would be monthlySalary

Row selection (WHERE clause)

- Sometimes, we often need to restrict the rows that are retrieved.
- This can be achieved with the WHERE clause, which consists of the keyword WHERE followed by a search condition that specifies the rows to be retrieved.
- The five basic search conditions (or predicates using the ISO terminology) are as follows:
  - **Comparison:** Compare the value of one expression to the value of another expression.
  - **Range**: Test whether the value of an expression falls within a specified range of values.
  - **Set membership**: Test whether the value of an expression equals one of a set of values.
  - **Pattern match**: Test whether a string matches a specified pattern.
  - **Null**: Test whether a column has a null (unknown) value.

Comparison search condition

- ***Example 5: List all staff with a salary greater than Kshs. 10,000.***
  - **SELECT staffNo, fName, lName, position, salary FROM Staff WHERE salary >10000;**
- Here, the table is Staff and the predicate is salary >10000.
- The selection creates a new table containing only those Staff rows with a salary greater than Kshs. 10,000.
- The rules for evaluating a conditional expression are:
  - An expression is evaluated left to right;
  - Sub expressions in brackets are evaluated first;
  - NOTs are evaluated before ANDs and ORs;
  - ANDs are evaluated before ORs.

Compound Comparison Search Condition

- ***Example 6: List the addresses of all branch offices in Nairobi or Nakuru.***
  - **SELECT * FROM Branch WHERE city =‘Nairobi’ OR city =‘Nakuru’;**
- In this example the logical operator OR is used in the WHERE clause to find the branches in Nairobi (city -‘Nairobi’) or in Nakuru (city =‘Nakuru’).

Range search condition (BETWEEN/NOT BETWEEN)

- *Example 7: List all staff with a salary between sh.20,000 and sh.30,000.*
  - **SELECT staffNo, fName, lName, position, salary**

    FROM Staff

    WHERE salary BETWEEN 20000 AND 30000;
- The BETWEEN test includes the endpoints of the range, so any members of staff with a salary of sh.20,000 or sh.30,000 would be included in the result.
- There is also a negated version of the range test (NOT BETWEEN) that checks for values outside the range.
- We could have expressed the above query as:
  - **SELECT staffNo, fName, lName, position, salary FROM Staff WHERE salary >=20000 AND salary <=30000**
- However, the BETWEEN test is a simpler way to express a search condition when considering a range of values.

Set membership search condition (IN/NOT IN)

- Example 8: List all managers and supervisors.
  - **SELECT staffNo, fName, lName, position FROM Staff WHERE position IN (‘Manager’, ‘Supervisor’);**
- The set membership test (IN) tests whether a data value matches one of a list of values, in this case either ‘Manager’ or ‘Supervisor’.
- We could have expressed the above query as:
  - **SELECT staffNo, fName, lName, position FROM Staff WHERE position =‘Manager’ OR position =‘Supervisor’;**
- However, the IN test provides a more efficient way of expressing the search condition, particularly if the set contains many values

Pattern match search condition(LIKE/NOT LIKE)

- **Example 9: Find all owners with the string 'Kisumu' in their address.**
- For this query, we must search for the string 'Kisumu' appearing somewhere within the address column of the PrivateOwner table.
- SQL has two special pattern-matching symbols:
  - % percent character represents any sequence of zero or more characters (wildcard).
  - _ underscore character represents any single character.
- All other characters in the pattern represent themselves.
- For example:
  - Address LIKE 'H%' means the first character must be H, but the rest of the string can be anything.
  - Address LIKE 'H_ _ _' means that there must be exactly four characters in the string, the first of which must be an H.
  - Address LIKE '%e' means any sequence of characters, of length at least 1, with the last character an e.
  - Address LIKE "%Kisumu%' means a sequence of characters of any length containing Kisumu.
  - Address NOT LIKE 'H%' means the first character cannot be an H.

Pattern Matching

- Using the pattern-matching search condition of SQL, we can find all owners with the string ‘Nakuru’ in their address using the following query.
  - SELECT ownerNo, fName, lName, address, telNo

    FROM PrivateOwner

    WHERE address LIKE ‘%Nakuru%’;
- **Note**, some RDBMSs, such as Microsoft Office Access, use the wildcard characters * and ? instead of % and _ .

NULL search condition (IS NULL/IS NOT NULL)

- ***Example 10: List the details of all viewings on property PG4 where a comment has not been supplied.***
- A null comment is considered to have an unknown value, so we cannot test whether it is equal or not equal to another string.
- If we tried to execute the SELECT statement using either of these compound conditions, we would get an empty result table.
- Instead, we have to test for null explicitly using the special keyword IS **NULL**:
  - SELECT clientNo, viewDate FROM Viewing

    WHERE propertyNo =PG4' AND comment **IS NULL**;

Sorting Results (ORDER BY Clause)

- In general, the rows of an SQL query result table are not arranged in any particular order (although some DB~~MSs may use a default ordering based, for example, on a primary key).~~
- However, we can ensure the results of a query are sorted using the ORDER BY clause in the SELECT statement.
- The ORDER BY clause consists of a list of column identifiers that the result is to be sorted on, separated by commas.
- A column identifier may be either a column name or a column number# that identifies an element of the SELECT list by its position within the list.
- The ORDER BY clause allows the retrieved rows to be ordered in ascending (ASC) or descending (DESC) order on any column or combination of columns, regardless of whether that column appears in the result.
- However, some dialects insist that the ORDER BY elements appear in the SELECT list. In either case, the ORDER BY clause must always be the last clause of the SELECT statement.

Single-Column Ordering

- ***Example 11: Produce a list of salaries for all staff, arranged in descending order of salary.***

  -
    ```
    SELECT staffNo, fName, lName, salary
    FROM Staff
    ORDER BY salary DESC;
    ```
- This is achieved by adding the ORDER BY clause to the end of the SELECT statement, specifying salary as the column to be sorted, and DESC to indicate that the order is to be descending.
- It is possible to include more than one element in the ORDER BY clause.

Using the SQL Aggregate Functions

- As well as retrieving rows and columns from the database, we often want to perform some form of summation or aggregation of data, similar to the totals at the bottom of a report.
- The ISO standard defines five aggregate functions:
  - COUNT – returns the number of values in a specified column;
  - SUM – returns the sum of the values in a specified column;
  - AVG – returns the average of the values in a specified column;
  - MIN – returns the smallest value in a specified column;
  - MAX – returns the largest value in a specified column.

Use of COUNT(*)

- *Example 12: How many properties cost more than sh. 350 per month to rent?*
  - SELECT COUNT(*) AS myCount

```
FROM PropertyForRent
WHERE rent > 350;
```

Use of COUNT (DISTINCT)

Example 13: How many different properties were viewed in May 2004?

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

Use of COUNT and SUM

- Example 14: *Find the total number of Managers and the sum of their salaries.*
  - SELECT COUNT(staffNo) AS myCount, SUM(salary) AS mySum

    FROM Staff WHERE position ='Manager';

Use of MIN, MAX, AVG

- *Example 15: Find the minimum, maximum, and average staff salary.*
  - SELECT MIN(salary) AS myMin, MAX(salary) AS myMax, AVG(salary) AS myAvg
  - FROM Staff;

Grouping Results (GROUP BY Clause)

- Use of GROUP BY
- ***Example 15: Find the number of staff working in each branch and the sum of their salaries***.

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

Restricting groupings (HAVING clause)

- The HAVING clause is designed for use with the GROUP BY clause to restrict the groups that appear in the final result table.
- Although similar in syntax, HAVING and WHERE serve different purposes.
- The WHERE clause filters individual rows going into the final result table, whereas HAVING filters groups going into the final result table.
- ***Example 16: For each branch office with more than one member of staff, find the number of staff working in each branch and the 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;

DATABASE UPDATES

- SQL is a complete data manipulation language that can be used for modifying the data in the database as well as querying the database.
- The commands for modifying the database are not as complex as the SELECT statement.
- In this section, we describe the three SQL statements that are available to modify the contents of the tables in the database:
  - INSERT – adds new rows of data to a table;
  - UPDATE – modifies existing data in a table;
  - DELETE – removes rows of data from a table.

Adding data to the database (INSERT)

- There are two forms of the INSERT statement.
- The first allows a single row to be inserted into a named table and has the following format:
  - **INSERT INTO TableName [(columnList)]**

    VALUES (dataValueList)

INSERT . . . VALUES

- ***Insert a new row into the 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 the Staff table supplying data for all mandatory columns: staffNo, fName, lName, position, salary, and branchNo.***
    - INSERT INTO Staff (staffNo, fName, lName, position, salary, branchNo) VALUES ('SG44', 'Anne', 'Jones', 'Assistant', 8100, 'B003');

Modifying data in the database (UPDATE)

- The UPDATE statement allows the contents of existing rows in a named table to be changed. The format of the command is:

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

- TableName can be the name of a base table or an updatable view.
- The SET clause specifies the names of one or more columns that are to be updated.
- The WHERE clause is optional; if omitted, the named columns are updated for all rows in the table.
- If a WHERE clause is specified, only those rows that satisfy the searchCondition are updated.
- The new dataValue(s) must be compatible with the data type(s) for the corresponding column(s).

UPDATE All Rows

- Give all staff a 3% pay increase.

```
UPDATE Staff
SET salary = salary*1.03;
```

- As the update applies to all rows in the Staff table, the WHERE clause is omitted.

UPDATE specific rows

- Give all Managers a 5% pay increase.

```
UPDATE Staff SET salary = salary*1.05
WHERE position = ‘Manager’;
```

- The WHERE clause finds the rows that contain data for Managers and the update salary = salary*1.05 is applied only to these particular rows.

UPDATE Multiple Columns

- Promote David Ford (staffNo = ‘SG14’) to Manager and change his salary to sh.18,000.

```
UPDATE Staff
SET position = ‘Manager’, salary = 18000
WHERE staffNo = ‘SG14’
```

Deleting Data From The Database (DELETE)

- The DELETE statement allows rows to be deleted from a named table.
- The format of the command is:

```
DELETE FROM TableName
[WHERE searchCondition]
```

- As with the INSERT and UPDATE statements, TableName can be the name of a base table or an updatable view.
- The searchCondition is optional; if omitted, all rows are deleted from the table.
- This does not delete the table itself – to delete the table contents and the table definition, the DROP TABLE statement must be used instead.
- If a searchCondition is specified, only those rows that satisfy the condition are deleted.

DELETE Specific Rows

- ***Delete all viewings that relate to property PG4.***

```
DELETE FROM Viewing
WHERE propertyNo = 'PG4';
```

- The WHERE clause finds the rows for property PG4 and the delete operation is applied only to these particular rows.

DELETE all rows

- Delete all rows from the Viewing table.
  - **DELETE FROM Viewing;**
- No WHERE clause has been specified, so the delete operation applies to all rows in the table.
- This removes all rows from the table leaving only the table definition, so that we are still able to insert data into the table at a later stage.

DATA DEFINITION LANGUAGE

- The main SQL data definition language statements are:
- CREATE SCHEMA
- CREATE DOMAIN
- ALTER DOMAIN
- CREATE TABLE
- ALTER TABLE
- CREATE VIEW
- CREATE INDEX
- DROP SCHEMA
- DROP DOMAIN
- DROP VIEW
- DROP TABLE
- DROP INDEX

DATA DEFINITION LANGUAGE

- Integrity Enhancement Feature
- **Required Data**
  - Some columns must contain a valid value; they are not allowed to contain nulls.
  - A null is distinct from blank or zero, and is used to represent data that is either not available, missing, or not applicable.
    - ***position VARCHAR(10) NOT NULL***
- **Domain Constraints**
- Every column has a domain, in other words a set of legal values.
- For example, the sex of a member of staff is either 'M' or 'F', so the domain of the column sex of the Staff table is a single character string consisting of either 'M' or 'F'
    - ***ex CHAR NOT NULL CHECK (sex IN ('M', 'F'))***

Entity Integrity

- The primary key of a table must contain a unique, non-null value for each row.
- For example, each row of the PropertyForRent table has a unique value for the property number propertyNo, which uniquely identifies the property represented by that row.

- The ISO standard supports entity integrity with the PRIMARY KEY clause in the CREATE and ALTER TABLE statements.
- For example, to define the primary key of the PropertyForRent table, we include the clause:
  - **PRIMARY KEY(propertyNo)**
- For example, to define the primary key of the Viewing table, which consists of the columns clientNo and propertyNo.
- We include the clause:
  - **PRIMARY KEY(clientNo, propertyNo)**

Referential Integrity

- A foreign key is a column, or set of columns, that links each row in the child table containing the foreign key to the row of the parent table containing the matching candidate key value.
- For example, to define the foreign key branchNo of the PropertyForRent table, we include the clause:
  - **FOREIGN KEY(branchNo) REFERENCES Branch**

Creating a Table (CREATE TABLE)

- CREATE TABLE TableName
- {(columName dataType [NOT NULL] [UNIQUE] [DEFAULT defaultOption] [CHECK (searchCondition)] [, . . .]}
- [PRIMARY KEY (listOfColumns),]
- {[UNIQUE (listOfColumns)] [, . . .]}
- {[FOREIGN KEY (listOfForeignKeyColumns) REFERENCES ParentTableName [(listOfCandidateKeyColumns)] [MATCH {PARTIAL FULL}
- [ON UPDATE referentialAction]
- [ON DELETE referentialAction]] [, . . .]}
- {[CHECK (searchCondition)] [, . . .]})

Creating a Table

- CREATE TABLE PRODUCT (
  - P_CODE VARCHAR(10) NOT NULL UNIQUE,
  - P_DESCRIPT VARCHAR(35) NOT NULL,
  - P_INDATE DATE NOT NULL,
  - P_QOH SMALLINT NOT NULL,
  - P_MIN SMALLINT NOT NULL,
  - P_PRICE NUMBER(8,2) NOT NULL,
  - P_DISCOUNT NUMBER(5,2) NOT NULL,
  - V_CODE INTEGER,
- PRIMARY KEY (P_CODE),
- FOREIGN KEY (V_CODE) REFERENCES VENDOR ON UPDATE CASCADE);

Changing a Table Definition (ALTER TABLE)

- The ISO standard provides an ALTER TABLE statement for changing the structure of a table once it has been created.
- The definition of the ALTER TABLE statement in the ISO standard consists of six options to:
  - 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.

Changing a Table Definition (ALTER TABLE)

- The basic format of the statement is:
  - ALTER TABLE TableName
  - [ADD [COLUMN] columnName dataType [NOT NULL] [UNIQUE]
  - [DEFAULT defaultOption] [CHECK (searchCondition)]]
  - [DROP [COLUMN] columnName [RESTRICT | CASCADE]]
  - [ADD [CONSTRAINT [ConstraintName]] tableConstraintDefinition]
  - [DROP CONSTRAINT ConstraintName [RESTRICT | CASCADE]]
  - [ALTER [COLUMN] SET DEFAULT defaultOption]
  - [ALTER [COLUMN] DROP DEFAULT]

ALTER TABLE

- Change the Staff table by removing the default of 'Assistant' for the position column and setting the default for the sex column to female ('F').
  - **ALTER TABLE Staff ALTER position**
  - **DROP DEFAULT;**
  - **ALTER TABLE Staff ALTER**
  - **sex SET DEFAULT 'F';**
- Change the PropertyForRent table by removing the constraint that staff are not allowed to handle more than 100 properties at a time. Change the Client table by adding a new column representing the preferred number of rooms.
  - **ALTER TABLE PropertyForRent**
  - **DROP CONSTRAINT StaffNotHandlingTooMuch;**
  - **ALTER TABLE Client**
  - **ADD prefNoRooms PropertyRooms**

END