# Definitions and Terms:

* Primary Key: Something that uniquely identifies a record in a table
* Foreign Key: Something that uniquely identifies a record in a different table
* Record: Row in a table
* Constraint: Rules for a data in a table
  + Not null
  + Default: Provides a default value of none
  + Unique
  + Foreign Key
  + Primary Key
  + Check: Test to see values of a field
  + Index
* Integrity: A concept and process to ensure that data is accurate, complete and consistent
  + Entity Integrity: No duplicate records in a database
  + Domain Integrity: Every field in every table must be of correct type, format, range
  + Referential Integrity: Record can not be deleted if it is being referenced in another table
  + User-Defined Integrity: Integrity Constraints that a user defines over tables and records
* RDBMS: Relational database management system
* Normalisation: Organising data in a database to be logically stored and efficiently stored to ensure there is no problems accessing data in the future
  + First Normal Form(1NF): Ensure there is no redundant/repeating data by forcing a primary key
  + Second Normal Form (2NF): Remove all partial dependencies of any field/column on the primary key. Assumes 1NF
  + Third Normal Form(3NF): Ensures each column is only dependant on the primary key. Assumes 2NF (and 1NF)
* Dependencies: Relationship between two attributes in a DB in which one attribute determines value of another attribute
  + Partial Dependencies: Dependant Attributes are partially determined by determinant attributes
  + Transitive dependency: Dependant Attributes are determined by a set of attributes not included in determinant attribute
* Data Definition Language (DDL): Statements for creating/deleting a database  
  and its tables
* Data Manipulation Language (DML): Statements for adding/retrieving data from tables
* Data Control Language (DCL): Statements which grant/remove privileges for a user
* Transactional Control Language: Statements for maintaining the state of the DB
* View: Selective static perspective on a database table. For all intents and purposes, acts and can be queried like a regular table. Columns outside of this must accept NULL values

Datatypes:

* + Bigint
  + Int
  + Smallint
  + Tinyint
  + Bit
  + Decimal
  + Numeric
  + Float
  + Real
  + Date/datetime
  + Smalldatetime
  + Time
  + Char
  + Varchar
  + varchar(max)
  + text
  + nchar
  + nvarchar
  + nvarchar(max)
  + ntext
  + binary
  + varbinary
  + varbinary(max)
  + image
  + sql\_variant
  + timestamp
  + uniqueidentifier
  + xml
  + cursor
  + table

# SQL Commands:

* Create an SQL DB:  
  CREATE DATABASE name;  
  use name;
* Create an SQL Table:  
  CREATE TABLE table\_name(  
  column1 datatype PRIMARY KEY Auto\_increment,  
  column2 datatype,  
  …);
* Creating an index for tables  
  CREATE UNIQUE INDEX index\_name  
  ON table\_name (col1, col2,….);
* Dropping a table:   
  DROP TABLE name;
* To drop an index from a table:  
  ALTER TABLE name,  
  DROP INDEX index\_name;
* Add a column:  
  ALTER TABLE name,  
  ADD col\_name datatype;
* Drop existing column:  
  ALTER TABLE name,  
  DROP COLUMN col\_name;
* Rename existing table:  
  ALTER TABLE name,  
  RENAME TO new\_name;
* Changing Data type of table:  
  ALTER TABLE name,  
  MODIFY COLUMN col\_name datatype;
* Insert row:  
  INSERT INTO table\_name(col1,…colN)  
  VALUES (val1,…valN);
* To change content of a row:  
  UPDATE table\_name  
  SET col1=val1,…colN=ValN  
  WHERE condition;
* To use a Boolean Criteria:  
  WHERE {Bool Condition} (note you can stack conditions) (!= is <>)
* To delete a row which match a condition:  
  DELETE FROM name,  
  WHERE condition
* To get specific fields:  
  SELECT (DISTINCT) col1,…colN  
  FROM table;  
  WHERE conditions
  + Distinct filters for uniqueness
* One can also query based off specific values using WHERE:  
  WHERE col\_name IN (val1,…valN);
* To mark a new state and commit  
  COMMIT;  
  SAVEPOINT save\_name
* To rollback:  
  ROLLBACK to [save\_name];
* Cartesian Join (Can join multiple versions of same table):  
  SELECT Col1, Col2…,  
  FROM table1,table2  
  WHERE {condition}
* To use inner join: (Only overlapping items)

SELECT Col1, Col2…,  
FROM table1 LEFT JOIN table2  
ON table1.col= table2.col

* To use left join: (Table 1 and overlap with Table 2 only)

SELECT Col1, Col2…,  
FROM table1 LEFT JOIN table2  
ON table1.col= table2.col

* To use right join: (Table 2 and overlap with table 1)

SELECT Col1, Col2…,  
FROM table1 RIGHT JOIN table2   
ON table1.col= table2.col

* To use full outer join: (Both tables and overlap)

SELECT Col1, Col2…,  
FROM table1 FULL OUTER JOIN table2  
ON table1.col= table2.col

* To join two query results:  
  SELECT col1 FROM table1   
  UNION   
  SELECT col2 FROM table2;
* To order use ORDER BY col1,col2
* Sub queries take place in the Where Clause (can create a new query and use it as a condition)
  + Can also apply SELECT from these
* To compare strings:  
  SELECT col1 from table1  
  WHERE col2 like ‘%string%’
* To select values and copy it into a new column  
  SELECT \* INTO new\_table FROM table1
* Can use aggregates in SELECT  
  SELECT agg(col1)
* After aggregating, we can condition on the aggregate using HAVING
* To group by column:  
  GROUP BY col ASC/DESC
* To use an alias, use the keyword as in form:  
  col1 as alias
* To create a view:  
  CREATE VIEW view\_name AS {query}
* NOT IN allows in the where clause allows us to get fields which aren’t located in the output of another query
* Rules for views:
  + Select cant use distinct
  + Select cant use aggregate/set functions or set operators
  + Select cant use order by
  + From can’t be multiple tables
  + Where clause cant have sub queries
  + Having/group by cant be included
  + All non-null columns from base table must be included to use insert
* Use of views:
  + Structure data
  + Restrict access to the data so users only sees what they need to
  + Synthesize and Summarize data from different tables
  + Generate reports
* To aggregate and keep every row, we use OVER (PARTITION BY ):  
  SELECT columns,  
  AVG(agg\_column) OVER (PARTITION BY  
  group\_column) AS alias,  
  FROM tables;
* RANK() allows us to values in a Partitioned dataset:   
  SELECT columns,  
  RANK() OVER (PARTITION BY  
  group\_column ORDER BY colum\_to\_rank) AS rank,  
  FROM tables;
* COALESCE() is commonly used alongside join to replace any null values formed from the joining to something else.  
  SELECT COALESCE(col1,value to be added instead of null), col2
* CONCAT allows us to modify strings in the select to form strings in the output  
  SELECT CONCAT(col1, “ (”, col2, “)”),
* GROUP\_CONCAT(col1 ORDER BY col1 SEPARATOR x) concats all values in the column together, orders it by a column and separates it by x.
* CASE statements are additional conditional statements one can use in queries  
  (CASE  
  WHEN condition\_1 THEN result\_1  
  WHEN condition\_2 THEN result\_2  
  ...  
  WHEN condition\_N THEN result\_N  
  ELSE result  
  END)
  + Else is run if none of the conditions are met
  + Can be used to construct new columns in select
  + Can be used to order things in ORDER BY

# PL/SQL

* Declare a function, parameters and return values  
  CREATE FUNCTION function\_name(param1 datatype1…..) RETURNs datatype  
   var\_name datatype;  
   Subtype var\_name is type;  
   DETERMINISTIC (It means the function always returns the same output for the same input)
* After creating the function, we then begin to declare variables and functionality  
  BEGIN  
   var1 datatype  
   {add function functionality here such as getting data form tables, arithmetic etc}  
   DBMS\_OUTPUT.PUT\_LINE(‘What you want to output’);  
  END
* To use a loop:  
  <<loop\_name>>  
  FOR I in 1….n LOOP  
    
    
  END LOOP
* To call a function:  
  output=function(params)
* If no data is found, we can do the following:  
  WHEN no\_data\_found THEN  
  dbms\_output.put\_line(‘No record);
* Triggers are a routine that is activated by a change in the DBMS for example:  
  Create or Replace Trigger  
  show\_salary\_changes  
  Before Delete or Insert or Update  
  on Employees  
  For Each Row  
  When (New.ID > 0)  
  DECLARE sal\_delta number;  
  BEGIN  
  sal\_delta := :New.Salary - :Old.Salary;  
  dbms\_output.put\_line(‘Employee: ’  
  || :Old.Name || ‘ change ’ || sal\_delta);  
  END