Database and SQL Fundamentals

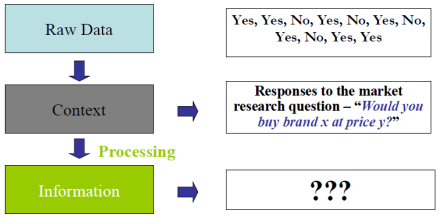
By Franco Diaz Licham

Contents

1 Data and Information

1.1 Data vs. Information

- **Data:** consists of raw fatcs not yet processed to reveal meaning to the end user. Building blocks of information and are usually stored in databases.
- **Information:** Produced by processing raw data to reveal meaning and thus requires context. It reveals the meaning of data to enables knowledge creation and making decisions. Should be acurate, relevant, and timely to enable decision making.

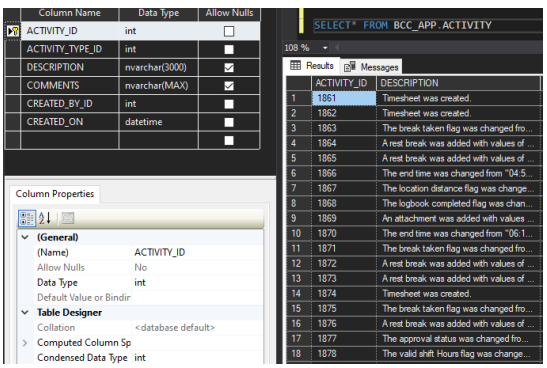


1.2 Database

- A shared, integrated structure that stores data. It stores two types of data user data and meta data.
- We require databases because we generate and consume lots of data every day. It is essential for businesses to survive and prosper. Collection, storage, aggregation, manipulation, dissemination, and management of data.
- Databases make data persistent and shareable in a secure way. Specialized structures that allow computer-based systems to store, manage, and retrieve data very quickly. Databases can be classified in various ways based on the context.

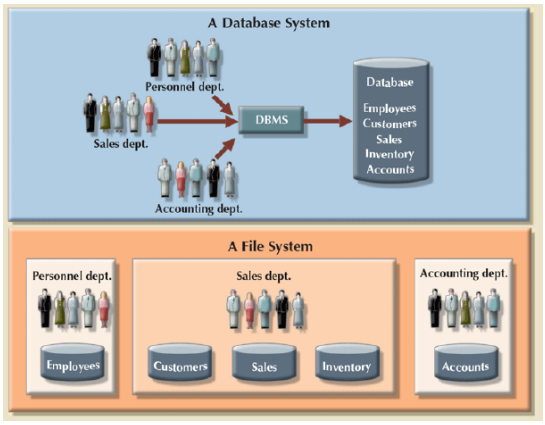
User Type	Single-user, Multi-user (Workgroup, Enterprise)
Location	Centralised, Distributed, Cloud
Data Usage	Operational (a.k.a. transactional or production), Analytical (Data Warehouse)
Data Type	General-purpose, Discipline-specific
Data Structure	Structured, Semi-structured, Unstructured
New Type	NoSQL (Non SQL), not the traditional database, NoSQL is the name given to a broad array of non-relational database to handle (e.g. social media on the Internet) - Unprecedented volume of data - Variety of data types and structures - Velocity of data operations

- **User data:** (raw facts) 1010, Larson, 0323, etc...
- **Meta-data:** (data about data) through which the end-user data is integrated and managed. It describes data characteristics (format, ec...) and relationships.

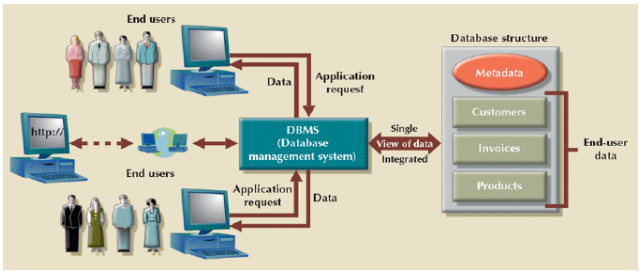


1.3 Data Systems

- **Manual systems:** Accomplished through a system of file folders and filing cabinets.
- **Computerized file systems** Data processing (DP) specialist created a computer-based system to track data and produce required reports. Hoever, it is difficult to get quick answers, complex administration, kack of security and limited data sharing and data is redundant.



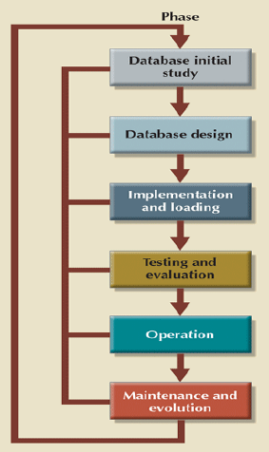
- **DBMS:** is a collection of all programs that manages database structures; controls access to the data stored in the databse; facilitates the sharing among multiple users and applications; it is an intermediate between user and the database; presents the end user with an integrated view of data; Provides a more efficient and effective data management and improves sharing, security, integration, access, decision-making and productivity. The DBMS receives all application requests and translates them into xomple operations required to fulfil those requests. The DBMS send back and answer to the application.



1.4 Database Life Cycle (DBLC)

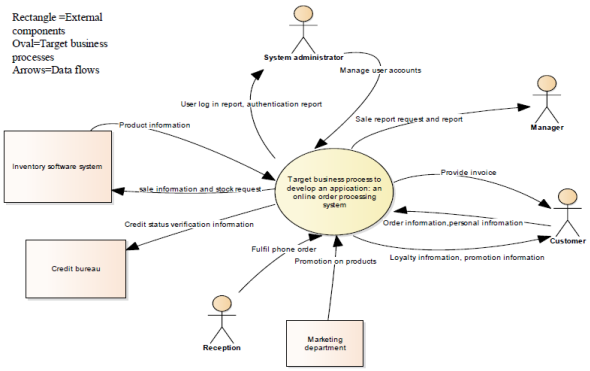
- A cycle that traces the history of a database within an organization's information system. DBLC contains six phases.
- **Database initial study:** understanding the data requirements. Analyze the company situation including operations, structure and objectives. Defines problems (problems in current system) and constraints (budget, hardware). It defines objectives (what must the new system do) and defines scope and boundaries.
- **Database design:** Create the conceptual design and choose DBMS software. Create the logical design including entities and relationships. Create the physical design. Use Modelling (developing a structure of DB using models). Models is an abstraction (simple representation) of complex real world objects.

- **Implementation and loading:** Install the DBMS. Create the database(s) and load or conver the data.
- **Testing and evaluation:** Test the databse. Fine-tune the database. Evaluate the database and its application program.
- **Operation:** Produce the required information flow.
- **Maintenance and evolution:** Introduce changes and make enhacements.



1.5 Data Flow Diagrams (DFD)

- shows flow of information (I/O) from the system. It helps to identify the data that the system has to store/maintain to meet the needs of external systems/users.



2 Database Design

2.1 Relational Database (RDB)

- Proposed by Edgar Frank Codd at IBM in 1970.
- Based on "relational" model which presents data as "relations" - a set of related tables.

2.2 Advantages vs. Disadvantages of RDB

- **Advantages:** Simple to use. Limits data redundancy. Maintains data intehrity and security. Security and access control. Offer logical/physical independence.
- **Disadvantages:** Requires good design of structure. Some systems can be expensive. Issue with handling unstructure data. Performance can be an issue.

2.3 Relational Model

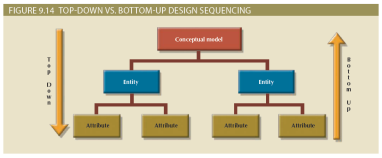
- **Entity:** person, place, thing or event abut which data will be collected and stored. E.g. Student, Course, Product, etc...
- **Attribute:** Characteristic of an entity. E.g. ID, Name, DoB, Address, etc...
- **Relationship:** association among entities. 1:1, 1:M, M:N, etc...
- **Contratins:** restriction placed on data whcih ensure data integrity. E.g. NOT NULL, Unquite, nvarchar(50), etc...

2.4 Database Design Steps

- **Conceptual design:** (ideation) design a database independent of database software and physical details. A Conceptual data model describes main data entities, attributes, relationships, and some constraints. Designed as software and hardware independent. Covers/captures business rules and system requirements (using ERD and normalisation). Use the business rules to determine parts of the model. Nouns many translate to entities or attributes. Verbs may translate to relationships. Conditions and requriements may translate to contraints. Use a DFD to bring all that information together.
- **Logical design:** (implementation) design an enterprise-wide database that is based on a specific DBMS software but independent of hardware-level details (user permissions, roles, etc...). Conceptual model is mapped to the specific format used by the selected DBMS. Logical Model is the database schema which includes system implementation, e.g., data types, columns, relationships, tables, etc... Validates logical model using normalisation, integrity contraints and against user requirements.
- **Physical design:** (configuration) physical storage, organization and access of database; ensures integrity, security, and performance. Define data storage organization (allocate space for database etc.). Define integrity and security measures (access right to the users based on their role). Determine performance measures (Fine tuning the database and queries).

2.5 Database design strategies

- **Top-down design:** starts by identifying the data sets and then defines the data elements for each of those sets. Involves the identification of different entity types and the definition of each entity's attributes
- **Bottom-up design:** first identifies the data elements (items) and then groups them together in data sets. First defines attributes, and then groups them to form entities.



2.6 Table

- A table contains data of related attributes of an entity. It consists of records (rows) and attributes (columns).
- A table is perceived as a two-dimensional structure composed of rows and columns. Each table row (tuple) represents a single entity occurrence within the entity set.
- Each table column represents an attribute, and each column has a distinct name. Each intersection of a row and column represents a single data value.
- All values in a column must conform to the same data format. Each column has a specific range of values known as the attribute domain.
- The order of the rows and columns is immaterial to the DBMS. Each table must have an attribute or combination of attributes that uniquely identifies each row.

2.7 Keys

- One or more attributes that determine other attributes.
- It is identitifed based on determination or "functional dependence". That is, if you know A you can know B. $A \longrightarrow B$.
- **Simple key:** If an attribute determined other attributes.
- **Composite key:** If a combination of attributes determines other attributes, that combination is a key.
- **Super key:** key that can uniquely determine any row in the table.
- **Candidate key:** Any minimal super key (super key without unnecessary attributes).
- **Primary key:** One of the candidate keys and chosen to be the unique row identifier.
- **Natural key:** A general accepted identifier for real world objects. Familiar to users. E.g. Tax File Number, Medicare Number, etc...
- **Surrogate key:** A primary key where numeric values only distinguish entities, and are normally generated by the DBMS and auto-incremented. E.g. ID. Very useful when a PK is composite and other tables reference this key. Synthetic key and artificial key are synonyms with surrogate key.

2.8 Integrity Rules

- **Entity Integrity:** Condition in which each row in the table has its own unique identity. All the values in the primary key column must be unique. No attribute in the primary key can contain a null.
- **Referential Integrity:** Foreign Keys (FK) are attribute(s) whose values match a candidate key (such as primary key) values in the related table. A FK must contain a value that refers to an existing valid row in another table.

2.9 Controlled Redundancy

- Redundancy is unnecessary duplication of data. Controlled redundancy makes a relational database work.
- Tables that share common attributes enable us to link

tables together.

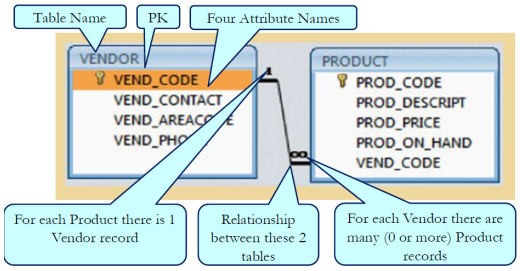
2.10 DBMS software selection

- **Cost:** including the original purchase price, maintenance, operational, licensing, installation, training, and conversion costs.
- **Features and tools:** Some database software includes a variety of tools that facilitate application development. For example, the availability of Query By Example (QBE), screen painters, report generators. Database administrator facilities, query facilities, ease of use, security, and third-party support also influence DBMS software selection.
- **Underling model:** can be relational, object/relational, or object-oriented or NoSQL.
- **Portability:** A DBMS can be portable across platforms, systems, and languages (e.g. SQLite).
- **DBMS hardware requirements:** Items to consider include processor(s), RAM, disk space, and so on.

3 Entity Relationship Diagrams

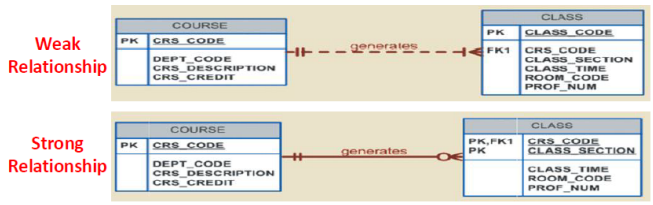
3.1 ERD

- Graphical representation of entity relational model in the conceptual design stage.
- Represents database structure in terms of entities (relation/table), attributes and relationships (primary and foreign keys).
- Visual modelling yields simplicity and is very effective for communication.
- Limited constraints representation (not all constraints are shown).
- There are various ERD notations including UML Notation, Crow's Foot Notation and Chen Notation.

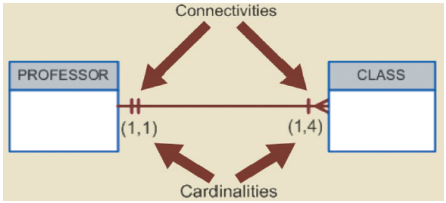


3.2 Crow's foot notation

- **Table:** An entity corresponds to a table and is represented by a rectangle containing the entity's name, PK and FK, and attributes.
- **Syntax:** Table name and attributes are written in capitals separate by an underscore and table names are singular nouns. Relationships are lowercase verbs.
- **Relationship:** an association between 2 entities by a solid (strong) or dashed (weak) line. weak occurs when primary key of the related entity does not contain a primary key component of the other entity. Strong occurs when Primary key of the related entity contains a primary key component of the other entity.



- **Connectivity:** (Ordinality) describes the relationship classification. That is, 1:1, 1:M and M:N. Closests to the table.
- **Participation:** how one row relates to others in a different entity. can be optional or mandatory. Optional is when one entity occurrence does not require a corresponding entity occurrence in a particular relationship. Mandatory One entity occurrence requires a corresponding entity occurrence in a particular relationship
- **Cardinality:** a property described by the connectivity and the level of participation between entities. It is the range (min,max) of entity occurrences (rows) associated with one occurrence (row) of the related entity. Farthest from the table. If not specified it takes the value of the connectivity and participation.
- For example, The cardinality (1,4) indicates that each professor teaches up to four classes, which means that the PROFESSOR table's primary key occurs at least once and no more than four times as foreign key values in the CLASS table. If the cardinality was (1,N), there would be no upper limit to the number of classes a professor might teach. Similarly, the cardinality (1,1) next to the PROFESSOR entity indicates that each class is taught by one and only one professor.



3.3 Relationship Degree

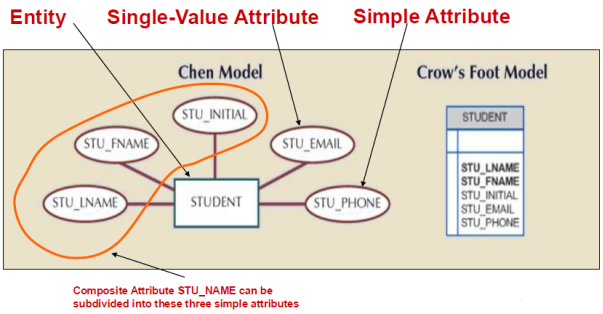
- The degree of a relationship is the number of entities associated with the relationship.
- **Unary Relationship:** (1 entity) one entity in a relationship with itself. Known as a recursive relationship.
- **Binary Relationship:** (2 entities) Two entities are associated in a relationship. Most common type of relationship.
- **Ternary Relationship:** (3 entities) An association among three different entities. These are not common.
- **Associative entity:** is used to implement an M:N relationship between two or more entities. Composed of the primary key attributes of each parent entity. May also contain additional attributes that play no role in connective process.

3.4 Database Schema

- A database schema is a way to logically group objects such as tables, views, stored procedures etc. Think of a schema as a container of objects.

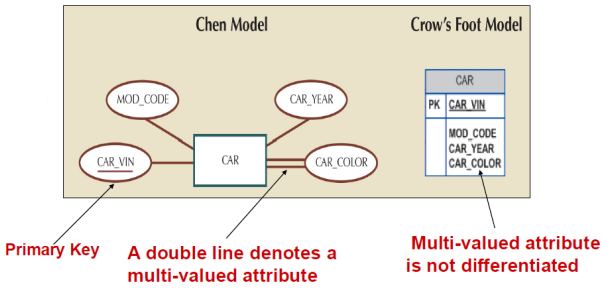
3.5 Attributes and Domain

- **Domain:** set of possible values for an attribute. E.g. States of Australia (NSW, ACT, etc...).
- **Required attribute:** must have a value (NOT NULL).
- **Optional attribute:** may be left empty (NULL is allowed).
- **Composite attribute:** can be further subdivided.
- **Simple attribute:** cannot be further subdivided.
- **Single-valued attribute:** can only have a single value at a particular instance of time. e.g., a person has one weight.



- **Multi-valued attribute:** can have many values, e.g., car color, or multiple contact numbers or several skills,

or several qualifications, and so on. These should not be used. Two possible solutions: (1) Create new attributes one for each of the original multivalued attribute's components. Split colours into body and top color attributes. (2) Create a new entity (associative entity) composed of original multi-valued attribute's components. Create two separate tables for body and top colours and use an associative entity to combine the two for a given car.



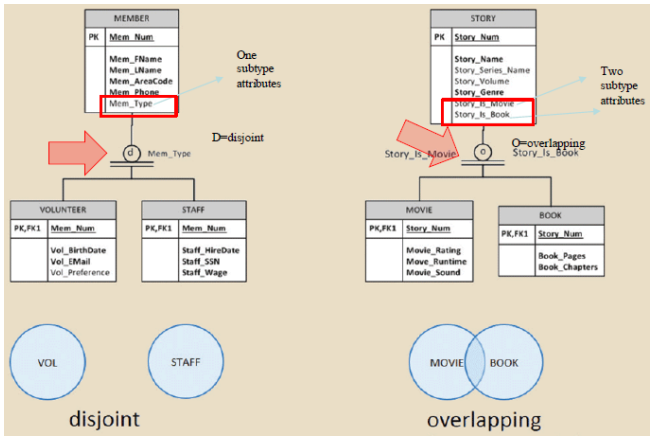
- **Derivable attribute:** An attribute whose value may be calculated (derived) from other attributes. No need to be physcially stored in the DB. Can be derived when needed. Use context to determine whether it is worth saving in the DB or to calculate it when required.

3.6 Existence Dependence vs Independence

- **Existence dependence:** (weak) Entity exists in the database only when it is associated with another related entity occurrence (row). Has a primary key that is partially or totally derived from parent entity in the relationship thus it has a strong relationships with another entity.
- **Existence independence:** (strong) Entity exists apart from all of its related entities. Referred to as a strong entity or regular entity.

3.7 Extended ER Model

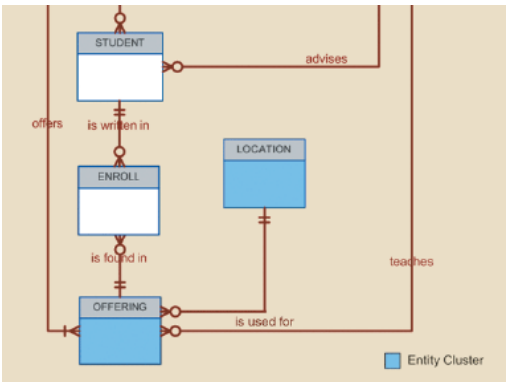
- Result of adding more semantic constructs to ER model as a result of modelling data requirements in complex real-world applications.
- Supertypes and Subtypes are parent and child entities respectively and the primary keys of supertype and subtype are always identical. Entity supertypes and subtypes are organized in a specialization hierarchy. Every subtype has one supertype to which it is directly related
- **Entity supertypes:** is an entity type that has got relationship (parent to child relationship) with one or more subtypes and it contains attributes that are common to its subtypes.
- **Entity subtypes:** are subgroups of the supertype entity and have unique attributes, but they will be different from each subtype.
- Use Extended ER Model when The different kinds of subtypes have each have unique attributes.
- **Descriminantor:** Define a special Supertype attribute known as the Subtype discriminator.
- **Disjoint:** contain a unique subset of the supertype entity set. Known as nonoverlapping subtypes. Implementation is based on the value of the subtype discriminator attribute in the supertype.
- **Overlapping:** contain nonunique subsets of the supertype entity set. Implementation requires the use of one discriminator attribute for each subtype.



- **Partial completeness:** not every supertype occurrence (row) is a member of a subtype.
- **Total completeness:** every supertype occurrence (row) must be a member of at least one subtypes.

TYPE	DISJOINT CONSTRAINT	OVERLAPPING CONSTRAINT
Partial	Supertype has optional subtypes. Subtype discriminator can be null. Subtype sets are unique.	Supertype has optional subtypes. Subtype discriminators can be null. Subtype sets are not unique.
Total	Every supertype occurrence is a member of only one subtype. Subtype discriminator cannot be null. Subtype sets are unique.	Every supertype occurrence is a member of at least one subtype. Subtype discriminators cannot be null. Subtype sets are not unique.

- **Specialization:** Top-down process. Identifies lower-level, more specific entity subtypes from a higher-level entity supertype. Based on grouping unique characteristics and relationships of the subtypes. Most commonly used.
- **Generalization:** Bottom-up process. Identifies a higher level, more generic entity supertype from lower-level entity subtypes. Based on grouping common characteristics and relationships of the subtypes
- **Entity clustering:** "Virtual/Abstract" entity type used to represent multiple entities and relationships in ERD. User to simply complex ERD (summarise many many tables). Formed by combining multiple interrelated entities into a single, abstract entity object. Avoid displaying attributes from the tables.



4 Normalisation

4.1 Anomalies

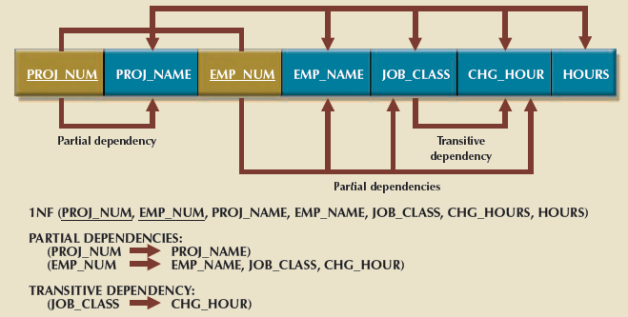
- Problems and issues that can occur because of poor database design include: Redundant data, Inconsistence and inaccurate data, Limitations to add/delete data. These occurs as a result of anomalies.
- **Insertion Anomalies:** occurs when certain attributes cannot be inserted into the database due to missing additional data. Caused by data unnecessary coupling of data. E.g. if a new employee is not assigned a department, their data cannot be inserted into the table if the department field does not allow null values
- **Update Anomalies:** Update anomalies occur when the same data is repeated in multiple rows, and changes are made in some but not all instances. E.g., if an employee's address changes and the update is made in one row but not in others, the database will contain inconsistent data.
- **Delete Anomalies:** Delete anomalies occur when deleting a certain row inadvertently leads to the deletion of other important data. E.g., if a department is shut down and all rows containing that department are deleted, the data of employees working solely in that department will also be deleted.

4.2 Normalisation

- process that evaluates and organises tables, attributes and relationships in a database in order to eliminate potential for data anomalies, unnecessary data dependencies and redundancies.
- divides entities and assign attributes to minimise data dependencies and redundancies.
- involves assessment of attributes in relations based on the concept of Functional Dependency.
- works through a series of stages to satisfy certain condition(s) called Normal Forms.
- a higher normal form is better than a lower normal form

4.3 Dependency

- **Functional Dependency:** A relationship between two attributes, where knowing the value of one attribute makes it possible to determine the value of another. For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B, represented as $A \longrightarrow B$. A is called *determinant* and B is called *dependent*. For any two tuples $t1$ and $t2$, if $t1$ and $t2$ have same values for attribute A, then they must have same values for attribute B.
- **Partial Dependency:** type of functional dependency in which the determinant is a part of the primary key. For example, if $(A, B) \longrightarrow (C, D)$, $B \longrightarrow C$, and (A, B) is the primary key, then the functional dependence $B \longrightarrow C$ is a partial dependency because only part of the primary key (B) is needed to determine the value of (C).
- **Transitive Dependency:** An attribute (not a PK or not a part of PK) is dependent on another attribute (determinant) that is not part of the primary key. More difficult to identify among a set of data. Occur only when a functional dependence exists among non-key attributes. Determinant and dependents both are non-key.



4.4 Normal Forms

- **1NF:** remove repeating groups (row must have single value). Each record must be unique and the table must have a PK.
- **2NF:** it is in 1NF. Ensures that there are no partial dependency. Encountered when PK is composite. Remove dependencies by moving each partial dependency into a new table with the key attribute of partial dependency from the original table as PK and their corresponding dependent attributes.

Player_ID	Item_Type	Item_Quantity	Player_Rating
jdog21	amulets	2	Intermediate
jdog21	rings	4	Intermediate
gila19	copper coins	18	Beginner
trev73	shields	3	Advanced
trev73	arrows	5	Advanced
trev73	copper coins	30	Advanced
trev73	rings	7	Advanced

Player_ID	Player_Rating
jdog21	Intermediate
gila19	Beginner
trev73	Advanced
tina42	Beginner

{ Player_ID } → { Player_Rating }

Player_ID	Item_Type	Item_Quantity
jdog21	amulets	2
jdog21	rings	4
gila19	copper coins	18
trev73	shields	3
trev73	arrows	5
trev73	copper coins	30
trev73	rings	7

{ Player_ID, Item_Type } → { Item_Quantity }

- **3NF:** It is in 2NF. Ensures that there are no transitive dependency. Remove dependent attributes from the table into a new table with the transitive determinant as the PK and leave the determinant as a FK in the original table.

Player_ID	Player_Rating	Player_Skill_Level
jdog21	Intermediate	4
gila19	Beginner	4
trev73	Advanced	8
tina42	Beginner	1

Player_ID	Player_Skill_Level
jdog21	4
gila19	4
trev73	8
tina42	1

Player_Skill_Level	Player_Rating
1	Beginner
2	Beginner
3	Beginner
4	Intermediate
5	Intermediate
6	Intermediate
7	Advanced
8	Advanced
9	Advanced

- **Boyce-Codd Normal Form:** Every determinant is a candidate key.
- **4NF:** It is in 3NF. Ensures that a table does not contain any multi-valued dependencies. Remove multi-valued

attributes from the table into new tables with the transitive determinant as the PK.

Model	Color	Style
Tweety	Yellow	Bungalow
Tweety	Yellow	Duplex
Tweety	Blue	Bungalow
Tweety	Blue	Duplex
Metro	Brown	High-Rise
Metro	Brown	Modular
Metro	Grey	High-Rise
Metro	Grey	Modular
Prairie	Brown	Bungalow
Prairie	Brown	Schoolhouse
Prairie	Beige	Bungalow
Prairie	Beige	Schoolhouse

Model	Color
Tweety	Yellow
Tweety	Blue
Metro	Brown
Metro	Grey
Prairie	Brown
Prairie	Beige

Model	Style
Tweety	Bungalow
Tweety	Duplex
Metro	High-Rise
Metro	Modular
Prairie	Bungalow
Prairie	Schoolhouse

- **5NF:** It is in 4NF. Decomposing a table into smaller tables to remove data redundancy and improve data integrity.

Person	Brand	Flavor
Jason	Frosty's	Vanilla
Jason	Frosty's	Chocolate
Jason	Alpine	Vanilla
Suzy	Alpine	Rum Raisin
Suzy	Ice Queen	Mint Chocolate Chip
Suzy	Ice Queen	Strawberry

Brand	Flavor
Frosty's	Vanilla
Frosty's	Strawberry
Frosty's	Mint Chocolate Chip
Alpine	Vanilla
Alpine	Rum Raisin
Ice Queen	Vanilla
Ice Queen	Strawberry
Ice Queen	Mint Chocolate Chip

Person	Brand
Jason	Frosty's
Jason	Alpine
Suzy	Alpine
Suzy	Ice Queen

Person	Flavor
Jason	Vanilla
Jason	Chocolate
Suzy	Rum Raisin
Suzy	Mint Chocolate Chip
Suzy	Strawberry

5 Structured Query Language SQL

5.1 Introduction

- A language used to implement and interact with DBMS
- Non-procedural language with less than 100 commands.
- Each SQL instruction is parsed, its syntax is checked, and it is executed one instruction at a time. All processing takes place at the server side.
- Many SQL dialects exists but the differences are minor
- Commands fall into the main categories of DDL, DML, TCL and DCL.
- There are 3 fundamental types of data character, numeric and date. There are other types such as blobs to store complex data such as images and videos.

5.2 Data Manipulation Language (DML)

5.2.1 Fundamental Operators

- Arithmetic operations can be used within DMLs. The precedence rule is: parentheses/brackets, then powers, then multiplications and divisions and then additions and subtractions.
- **SELECT:** selects attributes from rows in one or more tables or views. Whildcard character * used to select all records.
- **FROM:** specifies the tables from which data should be retrived. Can be tables or inner sub-queries.
- **WHERE:** restricts the selection of rows based on a conditional expression.
- **GROUP BY:** groups the selected rows based on one or more attributes.
- **HAVING:** restricts the selection of grouped rows based on a condition.
- **ORDER BY:** orders the selected rows based on one or more attributes. `ASC` or `DESC`
- **INSERT:** inserts rows into a table
- **UPDATE:** modifies an attribute's values in one or more table's rows.
- **DELETE:** removes one or more rows from a table.

```
-- select
SELECT col1 * col2, col3
    FROM table1
    WHERE col1 = value 1
    GROUP BY col2
    HAVING (col3 > value3 )
    ODER BY col3 DESC

-- update
UPDATE table1 SET col1 = value1
    WHERE col1 = value2

-- insert
INSERT INTO table1(col1, col2)
    VALUES(a, b)

INSERT INTO table1(col1, col2)
    SELECT col1, col2 FROM table2

-- delete
DELETE FROM table1
    WHERE col1 = value1
```

5.2.2 Comparison and logical Operators

- Used in conditional expressions
- **Comparison:** `=, <, >, <=, >=, <> or !=`
- **Logical:** `AND, OR, NOT`

```
SELECT col1
FROM table1
WHERE col2 > value1
AND col4 < value2
```

5.2.3 Special Operators

- Used in conditional expressions.
- Strings (characters-based data) comparison based on their numeric ASCII (American Standard Code for Information Interchange) codes
- **AS:** used to create a column aliases. This is optional.
- **BETWEEN:** checks whether an attribute value is within a range
- **IN:** Checks whether an attribute value matches any value within a value list
- **LIKE:** Checks whether an attribute value matches a given string pattern. `%` means any and all following or preceding characters are eligible and `_` means any one character may be substituted for the underscore. E.g. `%n` includes Johnson and Jernigan while `_o_es` includes Jones, Cones, Cokes, totes, and roles. Use `UPPER()` and `LOWER()` functions to account for capitalisation of words.
- **IS NULL:** Checks whether an attribute value is null
- **EXISTS:** Checks whether a subquery returns any rows
- **DISTINCT:** Limits values to unique values

```
-- operators
SELECT DISTINCT col1
FORM table 1
WHERE col2 BETWEEN value1 AND value2
AND col3 IN (value3, value4)
OR col4 IS NULL
AND col5 LIKE '%name'

-- subquery
SELECT col1, col2
FROM (
    SELECT *
    FROM table2
    WHERE col3 > value1
)
WHERE col1 = value2
```

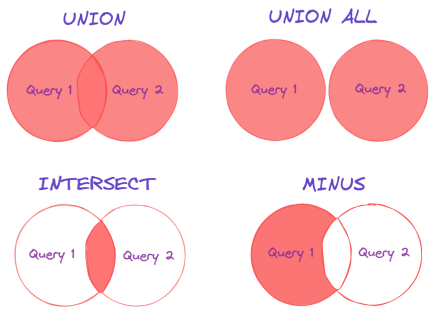
5.2.4 Aggregate Functions

- Used with the `SELECT` statement to return summaries on columns.
- use with the fundamental operator to summaries. E.g. `GROUP BY` will allow summary per group.
- **COUNT:** Returns the number of rows with non-null values for a given column
- **MIN:** Returns the minimum attribute value found in a given column
- **MAX:** Returns the maximum attribute value found in a given column
- **SUM:** Returns the sum of all values for a given column
- **AVG:** Returns the average of all values for a given column

```
-- count
SELECT COUNT(col2)
FROM table1

-- max
SELECT MAX(col1), MIN(col2)
FROM table1

-- sum
SELECT SUM(col1)
FROM table1
```

5.2.5 Relational Operators

- Relational operators are confine within the table to table level unless they are set operators.
- **PRODUCT:** `CROSS JOIN` in SQL, yields all possible pairs of rows from two tables
- **PROJECT:** the π operator, yields all rows for selected columns based on criteria.
- **SELECT:** the α operator, yields rows from all columns of a table based on criteria. Used with the * wildcard character. E.g. `SELECT * FROM product`.
- **JOIN:** combines data from two or more tables linked by common attributes.
- **Set Operators**
  - Special type of relational operators.
  - The set operations are confined within the query to query level. All queries must have the same type and number of columns for set operators to work.
  - Not all RDBMS will support all relational operators. E.g. MySQL does not support `DIFFERENCE`.
  - **UNION:** Combines all rows from two tables, excluding duplicate rows.
  - **INTERSECT:** Yields only the rows that appear in both tables.
  - **DIFFERENCE:** `MINUS` in SQL, yields all rows in the 1st table not found in the 2nd table, i.e., it subtracts/removes common rows from both from the 1st table

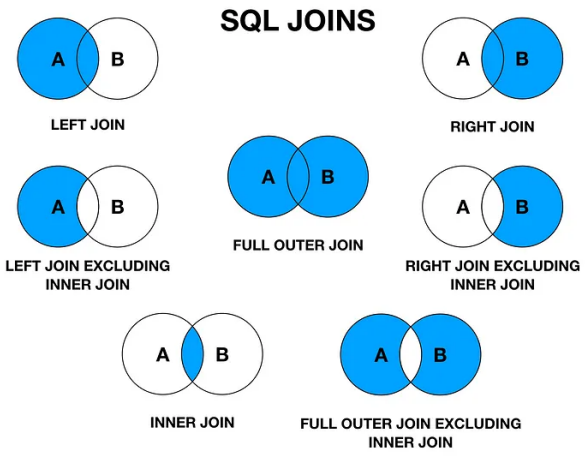



```
-- project (pi)
SELECT col1
FROM table1

-- select (alpha)
SELECT*
FROM table1

-- product
SELECT* FROM table 1
CROSS JOIN table2

-- union
SELECT col1 FROM table1
UNION
SELECT col2 FROM table2

-- intersect
SELECT col1 FROM table1
INTERSECT
SELECT col2 FROM table2

-- difference
SELECT col1 FROM table1
MINUS
SELECT col2 FROM table2
```

5.2.6 Types of JOIN

- There are two types: `INNER` and `OUTER` joins.
- There are combinations of these shown in the figure below.
- **INNER JOIN:** Returns records matching the condition(s) only from tables that are joined. There is no need to specify `INNER` as this is implicit of many databases.
  - **NATURAL JOIN:** links tables by selecting only the rows with the same domain(s) (values) and attribute names. It eliminates duplicate columns and if there are no common attributes, returns the relational product of the two tables, This is implicit and there is no need to specify in the `WHERE` clause. Unlike the Cartesian `PRODUCT`, which concatenates each row of the first table with every row of the second, natural join considers only pairs of rows with the same value of the same attributes in the tables.
  - **Equi join:** links tables on the basis of an equality condition that compares specified column(s) of each table. This is explicit, and there is the need to specify what attributes with the `ON` operator. Uses the = operator.
  - **Theta join:** links tables using an inequality comparison operator `<, >, <=, >=`.
  - **JOIN USING:** is another way of joining tables using attributes with the same name. Like natural join, eliminates duplicate columns but it throws an error if there are no common attributes.
  - **JOIN ON:** create an inner join, but the two tables do not have a commonly named attribute.
- **OUTER JOIN:** Matched pairs are retained and unmatched values in the other tables are left as NULL. There is no need to specify `OUTER` as this is implied.
  - **FULL OUTER JOIN (FULL JOIN):** yields all of the rows in both table, including those that do not have matching values, the columns that do not share common values are left NULL.
  - **LEFT OUTER JOIN (LEFT JOIN):** yields all of the rows in the first table, including those that do not have a matching value in the second table.
  - **RIGHT OUTER JOIN (RIGHT JOIN):** yields all of the rows in the second table, including those that do not have matching values in the first table.

```
-- natural join
SELECT *
```




```
FROM table1 NATURAL JOIN table2

-- equi join
SELECT col1, col2
FROM table1 JOIN table2
ON table1.col1 = table2.col2

-- theta join
SELECT *
FROM table1
JOIN table2 ON col1 > value2

-- join using
SELECT col1
FROM table1
JOIN table2 USING(col2)

-- full outer join
SELECT col1, col2
FROM table1
FULL OUTER JOIN table2
ON table1.col1 = table2.col1

-- left outer join
SELECT col1, col2
FROM table1
LEFT OUTER JOIN table2
ON table1.col1 = table2.col1

-- right outer join
SELECT col1, col2
FROM table1
RIGHT OUTER JOIN table2
ON table1.col1 = table2.col1

-- full outer join exlcuding inner join
SELECT col1, col2
FROM table1
FULL JOIN table2
ON table1.col1 = table2.col1
WHERE table1.col1 IS NULL
OR table2.col1 IS NULL

-- left join excluding inner join
SELECT col1, col2
FROM table1
LEFT JOIN table2
ON table1.col1 = table2.col1
WHERE table2.col1 IS NULL

-- right join exlcuding inner join
SELECT col1, col2
FROM table1
RIGHT JOIN table2
ON table1.col1 = table2.col1
WHERE table1.col1 IS NULL
```

5.3 Data Definition Language (DDL)

5.3.1 Fundamental Operators

- **CREATE TABLE:** Creates a new table in the user's database schema.
- **Constraints:** added to the column definitions. The RDBMS wil auto generate a name for the constraint unless speified. E.g. `CONTRAINT` name-of-constraint.
  - **NOT NULL:** Ensures that a column will not have null values.
  - **UNIQUE:** Ensures that a column will not have duplicate values
  - **PRIMARY KEY:** Defines a primary key for a table.
  - **FOREIGN KEY:** Defines a foreign key for a table. Uses the `REFERENCES` to establish the contraint to another table.
  - **DEFAULT:** Defines a default value for a column when no value is given.
  - **CHECK:** Validates data in an attribute
- Entity Integrity (PK) and Referential Integrity (FK) are applied automatically if constraints are defined.
- Define that happens to the the FK values when their corresponding in the primary table are updated or records are deleted.
- **Actions:**
  - **ON UPDATE:** defines what occurs to the FK when record is updated.
  - **ON DELETE:** defines what occurs to the FK when record is deleted.
  - **SET NULL:** sets corresponding value in the other table to null.
  - **CASCADE:** deletes or updates the value in both tables automatically.

```
CREATE TABLE table1 (
    col1 INT NOT NULL AUTO_INCREMENT,
    col2 DECIMAL(9,2) DEFAULT(0.00),
    col3 INT CHECK(col3 > value1),
    col4 INT NOT NULL,
    col5 INT AS (col3 + col4)
    PRIMARY KEY (col1),
    CONSTRAINT fk_col4_col3
        FOREIGN KEY (col4)
        REFERENCES table2(col3)
        ON DELETE CASCADE);
    CONTRAINT unique_col3
        UNIQUE(col3)
```

5.3.2 Other Operators

- **CREATE SCHEMA AUTHORIZATION:** Creates a database schema.
- **CREATE INDEX:** Creates an index for a table.
- **CREATE VIEW:** Creates a virtual table, subset of rows and columns, from one or more tables.
- **ALTER TABLE:** Modifies a table's definition (adds,

modifies, or deletes attributes or constraints). The command followed by a keyword that produces the specific change.
  - **ADD:** Adds a new column or a new contraint.
  - **MODIFY:** Modifies a column or a constraint.
  - **DROP:** Adds or removes a column.
- **CREATE STORED PROCEDURE:** Creates a stored procedure that can be executed by the EXECUTE command. Stored procedures are used for all DML commands and abide by the ownsership chain for execution.
- **CREATE TABLE AS:** Creates a new table based on a query in the user's database schema.
- **DROP TABLE:** Permanently deletes a table (and its data)
- **DROP INDEX:** Permanently deletes an index
- **DROP VIEW:** Permanently deletes a view
- **CREATE TRIGGER:** Permanently deletes a stored procedure

```
-- Adding new column
ALTER TABLE tablename
ADD (col1 VARCHAR(2))

-- adding FK constraint
ALTER TABLE table1
ADD FOREIGN KEY (fk_1)
REFERENCES table1

-- adding check constraint
ALTER TABLE table1
ADD CHECK(col2 >= value1) (ck_1)

-- modify a column
ALTER TABLE table1
MODIFY (col1 DECIMAL(9,2))

-- drop a column
ALTER TABLE table1
DROP COLUMN col1

-- create a view
CREATE VIEW view1
AS SELECT* FROM table1

-- alter view
ALTER VIEW view1
AS SELECT* FROM table1
WHERE table1.col1 > 1
```

5.4 Transaction Control Language (TCL)

5.4.1 Fundamental Operators

- COMMIT and ROLLBACK only work with DML commands that are used to INSERT, UPDATE, or DELETE table rows.
- Changes made using DDL commands (CREATE, ALTER, DROP) are not reverted
- **COMMIT:** Permanently saves data changes.
- **ROLLBACK:** Restores data to its original values from the last `COMMIT`.

5.5 Data Control Language (DCL)

5.5.1 Fundamental Operators

- **GRANT:** Gives a user permission to take a sytstem action or access a data object.
- **REVOKE:** Removes a previously granted permission from a user.

5.6 SQL in-built Functions

- SQL functions are very useful tools, similar to functions in programming languages.
- Functions always use numerical, date, or string values; a value may be part of a command or a table attribute.
- Some of SQL categories include:
- **Date and Time Functions:** All date functions take one parameter of a date or character data type and return a value (character, numeric or date type) Different implementation in different DBMS.
- **Numeric Functions:** Can be grouped in different ways, such as algebraic, trigonometric, and logarithmic
- **String Functions:** String manipulations - among the most-used functions in programming
- **Conversion Functions:** Allow you to take a value of a given data type and convert it to the equivalent value in another data type

5.7 Views

- Creates a virtual table, subset of rows and columns, from one or more tables and thus can be used as a table in SQL statements.
- These are used only for querying although soem RDBMS allow for updating records in views (not recommended).
- Views can be used to maintain users access control, provides data privacy and security by: restricting users to querying specific columns and rows of tables and limiting the degree of exposure of the underlying tables to the outer world by removing direct access to these tables.
- Querying to these tables is done via ownership chains where a view will be able to query into a table that the user has no permissions to query into only if the owner of the schema that contains the table is the same as the owner of the schema that contains the view. Since the vast majority of objects deployed in practice belong to the dbo schema, an ownership chain is almost always formed.
- Views take very little space to store; the database contains only the definition of a view, not a copy of all the data that it presents.
- Views are dynamically updated, i.e., re-created on demand each time it is invoked.
- Views may also be used as the basis for reports as they can join and aggregate information from multiple tables. Instead of executing complex SQL every time a particular report (e.g., weekly sales) is needed, a view can be defined with the required data.
- Treat a view as if it was a table and use the `SELECT` operator.
- **Updatable Views:**
  - Updtable views are views that can be used to update base tables that are used in the views. That is, it allows batch update of a base table based on data in another table.
  - If the (primary key) columns of the base table you want to update have unique values in the view, the base table is updatable as there is no ambiguity in the records.
  - Those DBMS that do support updatable views also

define that not all views are updatable. If the view defining SQL has Aggregate functions (e.g., AVG, COUNT, etc.), DISTINCT operator, GROUP BY clause, Set operators (e.g., UNION INTERSECTION, etc.), then the views are not updatable.
  - Since not all DBMS support updatable views then it is not recommended for updating records.

6 Advanced SQL

6.1 PL/SQL

- Procedural Language (PL) SQL.
- PL/SQL is ORACLE's proprietary procedural extension of SQL equivalent to T-SQL for Microsoft.
- Makes it possible to merge SQL and traditional programming constructs, such as variables, conditional processing (IF-THEN-ELSE), basic loops (FOR and WHILE loops), and error trapping, and to use and store procedural code and SQL statements within the database.
- PL/SQL performs a conditional or looping operation by isolating critical code and making all application programs be able call the shared code. Better maintenance and logic control.
- PL/SQL stores the code via the use Persistent Stored Modules.
- **Persistent Stored Module (PSM)** is a block of code that contains standard SQL statements and procedural extensions that is stored and executed at the DBMS server and represents business logic that can be encapsulated, stored, and shared among multiple database users.
- Users can use PL/SQL to create anonymous PL/SQL blocks, stored procedures, functions and triggers.

6.2 Anonymous PL/SQL Blocks

- Block of code that allows to use procedural constructs but it is not given a specific name
- Executes the block as soon as hit "Enter"
- Useful for testing purposes and to display specific message when an SQL is executed.
- These are not stored.
- **Assign operator (:=):** is the assignment operator as opposed to = which is the equality operator.
- **Statement declaration:** Each statement inside the PL/SQL code must end with a semicolon (;) and the whole block must finish with a (/). All PL/SQL That is, for PL/SQL blocks including anonymous, procedures, functions, packages and triggers, because they are multiple line programs, Oracle needs a way to know when to run the block, (;) indicates when a statement ends and a (/) is required at the end of each block to let Oracle run it.
- **The action:** The PL/SQL code enclosed between the `BEGIN` and `END` keywords.
- **%TYPE:** Inherits the data type from a variable that you declared previously or from an attribute of a database.

```
DECLARE variable1 INTEGER := 10;
BEGIN
    SELECT *
    FROM table1
    WHERE table1.col1 = variable1;
END;
/
```

6.3 Stored Procedures

- A named collection of procedural and SQL statements - stored in the database.
- Works via the ownership chaining for accessing tables.
- Stored procedure can be used to encapsulate SQL statements for a business activity that can be executed as a single transaction. E.g., you can create a stored procedure to represent a product sale, a credit update, or the addition of a new customer.
- They can substantially reduce network traffic and increase performance. Because the procedure is stored at the server, there is no transmission of individual SQL statements over the network, executed locally on the RDBMS.
- Reduce code duplication by means of code isolation and code sharing. creates unique PL/SQL modules that can be called by application programs and minimizes the chance of errors and the cost of application development and maintenance.
- Stored procedure can have zero or many arguments passed to it. These are called parameters.
- **IN and OUT** `IN, OUT, IN/OUT` indicates whether the parameter is for input, output, or both.
- **AS and IS** equivalent keyboards to begin a block.
- Data-type is one of the procedural SQL data types used in the RDBMS.
- The data types normally match those used in the RDBMS table creation statement.
- Variables can be declared between the keywords IS and BEGIN.
- You must specify the variable name, its data type, and (optionally) an initial value.
- **EXECUTE** used as a command to run the store procedure and pass the required parameters in SQL.
- There is no need to use `DECLARE` keyword when defining variables.

```
CREATE OR REPLACE PROCEDURE sp_1
param1 IN INTEGER
AS
BEGIN
    IF (param1 > value1)
    OR (param1 > value2) THEN
        UPDATE table1
        SET  col1 = value1
        WHERE table1.col1 = param1
    END IF;
END;
/
```

6.4 Triggers

- A trigger is a procedural SQL code that is automatically invoked by RDBMS when given data manipulation event occurs.
- It is associated with a database table; each table may have one or more triggers.
- It is executed as part of the transaction that triggered it. Triggers are critical to proper database operation and management.

- It is invoked before or after a data row is inserted, updated, or deleted.
- Triggers can be used to update table values, insert records in tables, and call other stored procedures.
- Triggers are used for (some examples): Creating audit logs, Automatic generation of derived column values, Enforcement of business or security constraints, – Creating replica tables for backup.
- **The triggering timing:** specified by the keywords `BEFORE` or `AFTER`. This timing indicates when the trigger's PL/SQL code executes—in this case before or after the triggering statement is completed.
- **The triggering event:** The statement that causes the trigger to execute (`INSERT`, `UPDATE`, or `DELETE`).
- **The triggering level:** There are two types
  - **A statement-level trigger:** default case if you omit the FOR EACH ROW keywords. This type of trigger is executed once, before or after the triggering statement is completed
  - **A row-level trigger:** This requires use of the `FOR EACH ROW` keywords. This type of trigger is executed once for each row affected by the triggering statement.

```
CREATE OR REPLACE TRIGGER trigger1
AFTER UPDATE OF col1 ON table1
BEGIN
    UPDATE table2
    SET table2.col1 = value2
    WHERE table1.col1 < value1
END;
/
```

6.5 Functions

- Built-in functions can be used only within SQL statements.
- PL/SQL functions are mainly invoked within PL/SQL programs (e.g., triggers and procedures).
- PL/SQL functions can be called within SQL statements provided that they conform to very specific rules that are dependent on DBMS environment.
- Functions must return a value.
- There is no need to use `DECLARE` when defining variables.

```
CREATE OR REPLACE
    FUNCTION func1(param1 INTEGER)
RETURN NUMBER IS
    value1 NUMBER := 0;
BEGIN
    SELECT SUM(col1) INTO value1
    FROM table1
    WHERE table1.col1 > param1
RETURN value1
END;
/
```

6.6 Cursors

- If the SQL statement returns more than one value, it will generate an error within a PL/SQL block.
- A cursor is a special construct used in PL/SQL to hold the data rows returned by an SQL query. A cursor can be thought as a reserved area of memory in which the output of the query is stored, like an array holding columns and rows. Cursors are held in a reserved memory area in the DBMS server, not in the client computer.
- **Implicit:** An implicit cursor is automatically created in PL/SQL when the SQL statement returns only one value.
- **Explicit:** An explicit cursor is created to hold the output of a SQL statement that may return two or more rows (but could return zero rows or only one).
- Once a cursor is declared, specific PL/SQL cursor processing commands can be used to access the information including OPEN, FETCH, and CLOSE anywhere between the BEGIN and END keywords of the PL/SQL block.
- **Commands**
  - **OPEN:** Opening the cursor executes the SQL command and populates the cursor with data. The cursor declaration command only reserves a named memory area for the cursor; it does not populate the cursor with the data. Before you can use a cursor, you need to open it.
  - **FETCH:** Once the cursor is opened, you can use the FETCH command to retrieve data from the cursor and copy it to the PL/SQL variables for processing. The PL/SQL variables used to hold the data must be declared in the DECLARE section and must have data types compatible with the columns retrieved by the SQL command. If the cursor's SQL statement returns five columns, there must be five PL/SQL variables to receive the data from the cursor. The first time you fetch a row from the cursor, the first row of data from the cursor is copied to the PL/SQL variables; the second time you fetch a row from the cursor, the second row of data is placed in the PL/SQL variables; and so on.
  - **CLOSE:** The CLOSE command closes the cursor for processing.
- **Attributes:**
  - **%ROWCOUNT:** Returns the number of rows fetched so far. If the cursor is not OPEN, it returns an error. If no FETCH has been done but the cursor is OPEN, it returns 0.
  - **%FOUND:** Returns a TRUE result if the last FETCH returned a row and FALSE if not. If the cursor is not OPEN, it returns an error. If no FETCH has been done, it contains NULL
  - **%NOTFOUND:** Returns a TRUE result if the last FETCH did not return any rows and FALSE if it did. If the cursor is not OPEN then it returns an error and if no FETCH has been done then it contains NULL.
  - **%ISOPEN:** Returns a TRUE result if the cursor is open (ready for processing) or FALSE if the cursor is closed. Remember, before you can use a cursor, you must open it.

```
CREATE OR REPLACE PROCEDURE sp1 IS
    param1 table1.col1%TYPE --(inherets type);
CURSOR cursor1 IS
```

```
    SELECT* FROM table2;
BEGIN
OPEN cursor1 ;
LOOP
    FETCH cursor1 ;
    EXIT WHEN cursor1%NOTFOUND;
END LOOP;
CLOSE cursor1;
END;
/
```