CST 8215 - Database

S Terai
Algonquin College of Applied Arts and Technology
Ottawa, Ontario, Canada

Edition 1.6-S
Fall 2019

Contents

Preface

The sequence of reading assignments are deliberate. For example, reading the summary of a chapter earlier than later, is suggested. The order of chapters covered in lessons are not sequential, this facilitates completion of labs with practice in SQL early in the course followed by theory on normalization. During lecture we will reference sections or subsections from other lessons. Treat this as a reference manual, workbook and a guide. It is difficult to read the owners manual of a new car from start to finish, at different times we need sections relevant to our needs, some this this analogy applies to this book.

Students are encouraged to write on this workbook, space is provided for answers. Sections written as *Aside:* provide additional thought, analogy or comparison to the idea discussed. The reader may defer reading these on the second pass; continuity is maintained if these sections are not read.

The agony and joy of creating fictitious names, I have been spared with; the pattern for person names is `Consonant, Vowel` ending with a consonant, or other similar pattern. Names are either five or three letters. A spreadsheet generates the names for me; a familiar name is coincidental. This document is typeset in LaTeX.

Feedback in improving this workbook including errors, I would welcome and gladly accept; please email me at, `terais@algonquincollege.com`.

Acknowledgments

Kumari Gurusamy has identified several corrections in the Spring 2015 edition. Sanaa Issa has used these notes for her labs and lectures, has helped in editing in the Fall 2017 Edition.

Lab 2 was first developed by Patricia Murphy and other instructors. It has been modified to work on databases that have changed over the years, in particular the user interface.

In an earlier writing assignment, Louisa Lambregts had provided valuable suggestions and comments, they are incorporated in this document.

He is eloquent, patient and knowledgeable. I am thankful that Mel Sanschagrin had agreed to review the document. All of his suggestions are indispensable.

Over the semesters, many students have identified errors and suggestions. They have been an invaluable source in improving this document.

In 2001 I had the opportunity to attend a seminar on *The ICE Approach* [14]. Many pedagogical ideas have been incorporated from their book in my teaching.

Part of this document was prepared during my sabbatical year while at Algonquin College.

Dedicated to students – past and present.
This document has been possible because of you.
You have been patient – Thank You.

To the Student

Consistent and deliberate practice will give you confidence and a sound understanding of database theory and its query language.

Make full use of your lab hours and use them wisely. Your learning is consolidated by *doing* the lab exercises and repeating them.

Aquaint yourself with at least one of the following study techniques. The link leads you to a short document. Alternatively you may refer to wikipedia or search the web for the techniques.

1. SQ3R - Details can be found at `http://www.wpi.edu/Images/CMS/ARC/SQ3R_At_A_Glance.pdf`.
   Lately SQ4R is proposed as an extension to SQ3R. The last R in 4R is for `wRite`
2. PQRST - `http://www.lethbridgecollege.net/elearningcafe/images/stories/pdf/pqrst.pdf`

The lab and lecture time is intended to be collaborative, not competitive. You will learn the topic in a different way by helping your fellow students. Do not hesitate to ask questions and get clarifications.

1

Database - An Introduction

1.1 Introduction

The core of any information system is a database. It is used by all computing applications, its size, structure and type vary. Formal knowledge of a database, especially relational database, is an important skill in your IT career.

This lesson begins introducing a few terms, differentiating between data and information. Data when processed is information, information when processed is knowledge.

1.2 Objective

- Appreciate the role of databases in commercial, industrial and government organizations
- Describe components of database management system (DBMS) and their interaction
- Identify skill categories required at the workplace to run a database
- Differentiate between *data* and *information*
- Determine storage and use of metadata

1.3 Terms

Entity A person, place, object, event or concept; an entity becomes relevant if an organization has a need to maintain data on the entity. [3, Page 538]

Relation Refer Section 5.3, page 60 of these notes.

Data Element A smallest item in a database. Loosely referred as a *field*.

Attribute The name and description for a data element. *Aside:* Strictly, in relational model, the term *attribute* is used instead of *field*. This difference is historical from the time the relational model was first proposed in 1969. All data elements in a *column* have the same characteristics of the data, these characteristics are referred to as attribute. Loosely speaking an attribute corresponds to a column of a relation.

Entity Relationship Model A *logical* representation entities and their relationships.

Entity Relationship Diagram (ERD) A physical representation of a logical model.

Relational Database A database that stores data in relations associated with relationships.

Relationship An association between one, two or three more entities. *Aside:* A relationship has a cardinality and a degree. In this course we discuss binary degrees, and to some extent unary degrees, we do not discuss ternary degrees.

Cardinality The number of ***instances*** one entity associates with another entity. Examples of cardinality are `one:one`, `one:many`, `many:many`. The word *instances* is important. You should be able to differentiate between the two terms, **cardinality** and **degree**.

Degree The number of entities that participate in a relationship. There are three degrees: unary, binary and ternary.

Systems Development Life Cycle (SDLC) The traditional method used to plan, analyze, design, implement and maintain an information system.

1.4 Types of Relationship

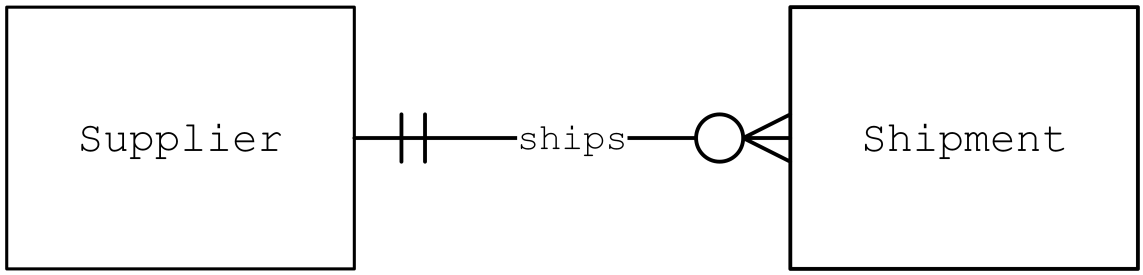


Figure 1.1: Supplier Shipment

In figure 1.4 a supplier can ship zero, one or many shipments. A shipment can be shipped by one and only one supplier. Stated differently, it is not possible for a shipment to arrive without a supplier name identified and a shipment cannot arrive from more than one supplier. The two bars on the Supplier side of the relationship is called mandatory one; i.e. must have one and only one.
The *degree* of the relationship is **binary**, it has two entities, `Supplier` and `Shipment`. Later we shall see two other degrees between entities. Refer figure 2.8 on page 27 for another example.

1.5 Learning Activities

Examples of small databases

List examples in daily life where you would use databases. Think about a hobby, small business, club activities or club membership, game statistics, travel. Choose an example that interests you and build it toward your assignment. Your application should have atleast five entities. As we explore ideas in the classroom write them down in the space provided below.

1.5.1 A Table and its Components

Data is stored in a table; a table has rows and columns. Similar to a spread sheet or a word processing document. Tables in a database have more meaning compared to the tables in a document. Even a small database has several tables which can interact with each other in storage and data retrieval.

Metadata Data that describes properties and characteristics of data is called metadata. Table 1.1 illustrates a small portion of metadata. Among other items metadata includes *a*) name of data element *b*) type, i.e. character, alphanumeric, date, decimal, boolean *c*) length *d*) minimum and maximum value, if applicable *e*) a short description *f*) the source of data, i.e. its origin. Examples of data sources are: customer, registrars office, accounting department.

Name	**Type**	**Len**	**Description**	**Source**
BookID	Fixed Character	13	Book ISBN	Assigned by publisher
Book Title	Variable Character	40	Title of the book	Usually given by author
AuthorID	Fixed Character	10	Surrogate Key	Assigned by system or by user
Author	Variable Character	30	Name of Author	Entered by user
BorrowerID	Fixed Character	7	Surrogate Key	Assigned by system or user defined
Borrower Name	Variable Character	30	Borrowers Name	Entered by user

Table 1.1: Metadata, an example

1.6 Data Model

A data model is a description of users data, relationship between data, constraints on data among other details. This model translates into Data Definition Language (DDL). A data model is an abstraction that focuses on the essential aspects of an information system. It aids in communicating users requirements to stakeholders. Stakeholders include developers, end users, administrators and project sponsors among others. There are several methods used in representing a data model. A common diagramming method is an Entity Relation Diagram (ERD). ERD's have several different notation, in this course we use `Crows Feet` Notation.

Entity Relation Diagram An Entity Relationship Diagram is pictorial representation of data. It consists of entities and relationships between the entities. At least two types of ERD's are used - **Logical** and **Physical** ERD's. They are used in different stages of the system development life cycle (SDLC). A Logical ERD is created at the initial stages of the SDLC. This ERD is independent of the database model that will be used. When creating this logical diagram the system analyst does not consider the database *model* that will be used to implement the system.
Aside: A database model is not the same a data model.

As the development progresses, the logical ERD is refined and transformed into a physical ERD. In our study the database model is a Relational Database. An apparent transformation is resolving a many-to-many relation into one-to-many relation. Other details added to physical ERD's for a relational database model is determining prime key, foreign key and constraint.

In most cases, there is a relation between two entities. In some cases an entity is related to itself, and in other cases it there are three entities that relate to each other.

Degree of a Relation When two entities are related in a binary relation, the degree is two. An entity related to itself it in a unary relation, i.e. of degree one. Three entities related in a ternary relation has a degree of three. In this course we shall work with entities of degree one and degree two only. We shall not study relationships of degree three.

Associative Entity An entity that is used to resolve a many-to-many `m:m` relationship. An associative entity is represented by rounded corners.
A Logical ERD may have two entities that are related by a `m:m` relationship. This relation cannot be implemented in a Relational Database. By introducing an associative entity an `m:m` relation is resolved into two `1:m` relations. An example of an associated entity in a Physical ERD is shown in figure A.5 on page 117.

Forward Engineering An database designer could model data using a data modeling tool. The tool then creates DDL from the model. Compare this processes to an architect drawing a structure and the diagramming tools lists the materials needed to build the structure.

Reverse Engineering Often a database schema already exists, created by DDL, a database designer wants a pictorial representation of the code. A data modeling tool can create an ERD from the DDL. In our course we shall first draw a logical ERD using pencil and paper, determine the prime key and foreign key. Next we shall write the DDL using the editor (pgAdmin3) test and run the DDL. Finally we will use the data modeling tool to reverse engineer model from the DDL. In one or two instances we shall forward engineer the model, most of the time it will be reverse engineering. We need to be able to write DDL and DML statements fluently. *Aside:* DML statements are not required to reverse engineer an ERD, only DDL statements are needed.

1.7 ERD Exercises

Draw a logical ERD from the rules given.

I A country can have many cities. A country may not have any cities. A city must belong to one and only one country.

II A vegetable can have many nutrients. A vegetable must have at least one nutrient. A nutrient can be obtained from more than one vegetable.

III A household may have taxpayers. A household may not have any taxpayers. A taxpayer must belong to a house.

IV A student can have only one UPass. A student may not have a UPass. A UPass must belong to only one student.

1.8 Drawing Physical ERD

First draw a logical ERD and then a physical ERD for the given abstract and business rules.

I. Registration. Student-Course

Rules A Student can enroll in one or more courses. If a student is registered, she is considered a student, even though she may not have enrolled in any courses. A course may have more than one student enrolled. A course that has been offered may not have any students enrolled in it.

II. A Trading Company

Abstract The trading company sells computer parts. The company maintains a list of customers with some details such as address and phone numbers. A list of products in the inventory is maintained. Customer purchases are initiated by creating an invoice. A purchase can have one or more products, an invoice can have one or more products of the same type.

Rules A customer can have at least one invoice. A customer may not have any invoice. An invoice must have one and only one customer. A product that is purchased is shown in an invoice. An invoice can have one or more products. A product can be purchased several times, i.e. a product can appear in one or more invoices. It is possible that a product may not be sold, i.e. it will not appear in a invoice at all.

1.9 Review Questions

1.9.1 Written Answers

1. Define the term *data*.

2. Explain what is meant by the term *database*.

3. Explain what is meant by a DBMS.

4. Name some database products and their vendors.

5. What is meant by *information*? Differentiate *information* from *data*.

6. Explain what is meant by *metadata*. Give two examples of data and metadata. What does metadata include? What does metadata not include?

1.9.2 Multiple Choice - Select one correct answer

Questions marked ™ are from TestGen by Prentice Hall

1. Software used to create, maintain, and provide controlled access to databases is called
   (a) Computer Aided Software Engineering (CASE) Tools
   (b) Graphical User Interface (GUI)
   (c) Database Management System (DBMS)
   (d) Network Operating System (NOS)
   (e) Computer Assisted Design (CAD)
2. All of the following are primary purposes of a database management system (DBMS) EXCEPT: ™
   (a) storing data
   (b) providing an integrated development environment
   (c) creating data
   (d) updating data
3. A relationship is an association between entity types
   (a) True
   (b) False
4. The _____ of a relationship is the number of entity types that participate in the relationship
   (a) attribute
   (b) cardinality
   (c) degree
   (d) constraint
5. A rule that specifies the *number of instances* of one or more entities
   (a) attribute
   (b) cardinality
   (c) degree
   (d) constraint
   (e) relationship

1.9.3 Short answer questions

1. Processed data is called _____
2. Data in a table is stored in the form of _____ and _____
3. Description of properties of data is called _____

1.10 Summary

The term DBMS indicates it is a system, i.e. a DBMS consists of several components, not all components may be available in a package. Web development has added an additional dimension to databases.

2

Introduction to SQL

2.1 Introduction

Our focus in this chapter is to get familiar with SQL and achieve a working knowledge of basic SQL commands and some intermediate and advanced commands. Several important SQL clauses are explained in this chapter, they will form the foundation of your SQL knowledge.

2.2 Objective

- Define and differentiate Data Definition Language (DDL), Data Manipulation Language (DML) and Data Control Language (DCL)
- Identify milestones in the history of `SQL`, the official standard for relational databases.
- Create a small database
- Establish relationships between the tables and write queries using tables
- Implement constraints that maintain database integrity
- Identify SQL 2008 standards and new commands

2.3 The SELECT Statement

General syntax of a basic SQL statement and the processing order is shown in figure 2.1. Note the three types of brackets that are used.

{ } - the braces indicate repeating values. One or more items of that type can be included in the command. In the example below `column` is within braces, it means one or more columns can be specified in the `SELECT` statement.

[ ] - square brackets indicate optional values.

< > angular brackets indicate mandatory values. Items contained within these brackets must be specified for the command to work correctly.

The vertical bar `|` implies that one item from the list is required. For example, in `ALL | DISTINCT` you need to specify only one of the two items, not both. *Note:* The `SELECT` clause accepts column name and expression.

General Syntax

A general syntax for the `SELECT` statement is:

```
SELECT [ ALL | DISTINCT ] {column / expression}
FROM {table}
[WHERE conditional expression]
[GROUP BY group_by_column_list]
[HAVING conditional expression]
[ORDER BY order_by_column_list]
```

Figure 2.1: SQL General Syntax. Ref: [2] Page 248.

2.3.1 Processing Order

Although the `SELECT` keyword appears first in the statement it is the parsed much later. Figure 2.2 illustrates the order of processing. The first statement to get evaluated in a SELECT statement is the `FROM` keyword, this is the reason aliases in `WHERE` and `HAVING` clauses are not permitted.

For example the `SQL` statement will flag an error,

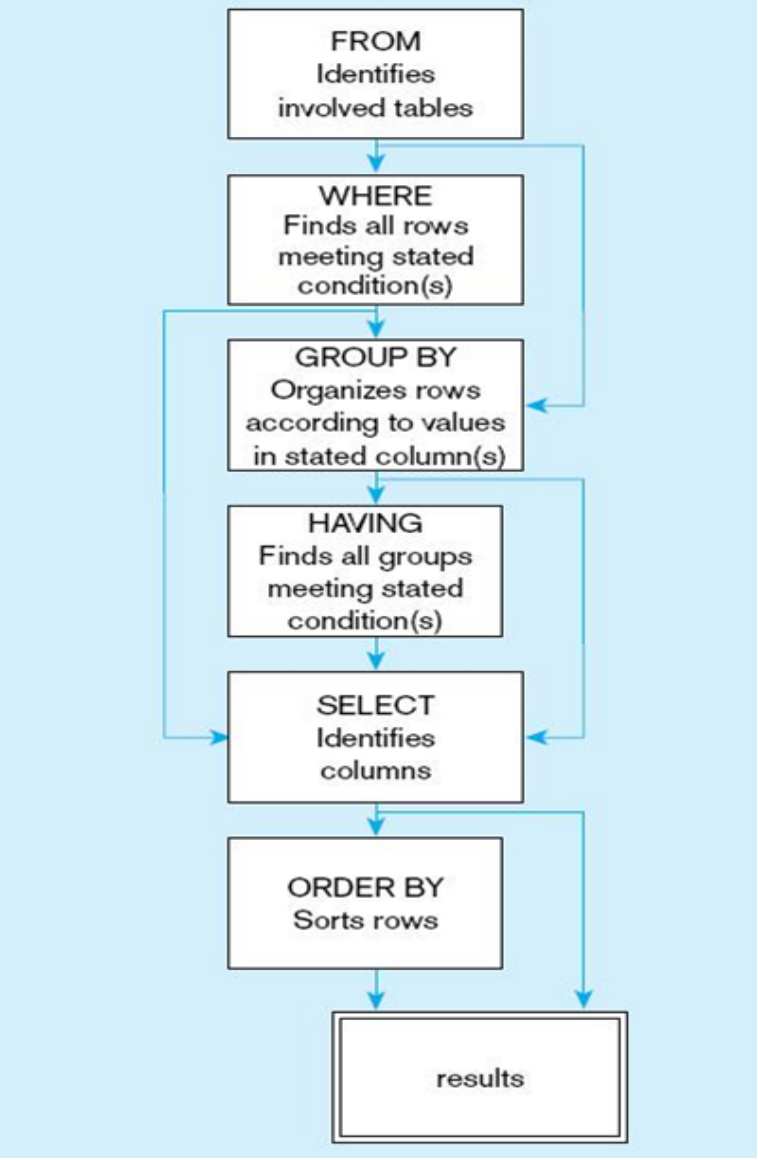


Figure 2.2: SQL Processing Order: Reference: [2]

2.4 DDL, DML, & DCL

> Any given data sublanguage is a combination of atleast two subordinate languages: a *data definition language* (DDL) ... and a *data manipulation language* (DML)...
> Ref. [10, CJ Date, Page 36]

DDL supports the definition or declaration of database objects, DML supports processing of the objects. Ref. [10, CJ Date, Page 3
Examples of DDL include `CREATE`, `DROP`, `CONSTRAINT`, `PRIMARY KEY`. Examples of DML include `INSERT`, `UPDATE`, `SELECT`.

Data Control Language (DCL) is a set of commands that are used to allow or deny access to a database system. Examples of DCL include `CREATE ROLE`, `DROP ROLE`, `GRANT` and `REVOKE`. Privileges may be granted or revoked to/from a user or a *role*.

Definition: A *role* is a set of permissions assigned by an administrator to users or groups.

Multiple Choice Questions.

Select the best answer. Each question has one correct answer.

1. Commands used to define a database, which includes creating, modifying and deleting tables, and establishing constraints.
   (a) Data Definition Language (DDL)
   (b) Data Manipulation Language (DML)
   (c) Data Control Language (DCL)
2. Commands used to maintain and query a database, which includes data insertion, deletion and modifications to data.
   (a) Data Definition Language (DDL)
   (b) Data Manipulation Language (DML)
   (c) Data Control Language (DCL)
3. Commands used to control a database, which includes access control.
   (a) Data Definition Language (DDL)
   (b) Data Manipulation Language (DML)
   (c) Data Control Language (DCL)
4. `CREATE <table>`, `CONSTRAINT <label> PRIMARY KEY <attribute>` are examples of
   (a) DDL
   (b) DML
   (c) DCL
5. `INSERT`, `UPDATE`, `SELECT` are examples of
   (a) DDL
   (b) DML
   (c) DCL
6. `GRANT`, `ADD`, `REVOKE` are examples of
   (a) DDL
   (b) DML
   (c) DCL
7. Which term best describes the definition:
   A set of permissions assigned by an administrator to users or groups
   (a) root
   (b) role
   (c) user
   (d) account holder
   (e) superuser

2.5 Access Control

Consider the two scenarios:

Scenario 1: You are mandated to have only one (physical) key, this key works for the front door to your house, garage, car, mailbox, office door, safe deposit box in the bank and two other assets you own.

Question: How would you adapt? Think of some scenarios where this system would be difficult in practice.

Scenario 2: A manufacturing organization has about 8,000 employees and several departments, sales, marketing, design, inventory, service, payroll, executive management, finance, public relations. These departments have their systems on a computer using postgres. All these departments have one account called `postgres` with the same password. Clearly, this system be difficult to manage even if all employees were honest, trustworthy and diligent.

At installation, PostgreSQL provides a login role called `postgres`. *Aside:* It also creates a database called `postgres`.

2.6 Case Sensitivity in PostgreSQL

Keeping with the tradition of Ingres, Postgres is case sensitive to object names; many databases are not case sensitive. Two examples shown below illustrate the differences. Forward engineering a schema will put quotes before and after an object name (tables, constraints...). The label you provide to aa modeling tool such as `Toad` will affect your code.
Aside: There is a way to configure Toad to generate code otherwise, this section considers the default setting.

Example 1. Using quotes to define objects Consider the following `SQL` code fragment.

```
CREATE TABLE "Customer_T" (
Cust_Id CHAR( 4 ),
Cust_Fname VARCHAR( 30 ) NULL, ...  ...
```

Figure 2.3: Using quotations to define an object

The table name `Customer_T` is within quotes, Postgres will preserve the object name you have provided.

The following `SELECT` statement will run on the `DML` for the code shown in figure 2.3
`SELECT * FROM "Customer_T";`

This statement will not run
`SELECT * FROM Customer_T;`

The error message shown below gives the insight to the inner working of postgreSQL.

```
ERROR:  relation "customer_t" does not exist
LINE 1: SELECT * FROM Customer_T;
^
ERROR: relation "customer_t" does not exist
SQL state: 42P01
```

Figure 2.4: SQL Case Sensitive Error

The error tells us the Postgres has converted the object name `Customer_T` to lower case, because there were no quotes around it. The code in figure 2.3 will need `DML` statements shown below; the object name must be in quotes.
`INSERT INTO "Customer_T" VALUES ('C001', ...  ...`

Example 2. Not using quotes to define objects Consider the following `SQL` code fragment.

```
CREATE TABLE Customer_T (
Cust_Id CHAR( 4 ),
Cust_Fname VARCHAR( 30 ) NULL, ...  ...
```

Figure 2.5: Defining an object without quotes

Compare figure 2.5 and figure 2.3. What is the difference?

For the DDL statements in figure 2.5 the following DML statements will run.

```
SELECT * FROM Customer_T;
SELECT * FROM customer_t;
SELECT * FROM CUSTOMER_T;
```

Aside: In a later assignment you will query the INFORMATION_SCHEMA; it provides information about tables, views, columns and procedures. Querying the INFORMATION_SCHEMA requires that tables names be in lower case, the way Postgres stores them internally. Refer page 147 figure B.2 to see how the table names are in typed lowercase.

2.7 Client-Server Architecture of a DBMS

Use the space below to draw a schematic diagram of a client-server architecture.

2.8 PostgreSQL Interface

PostgreSQL uses client-server architecture. Both the client and server are installed on your laptop. To access the server there are two client programs - `psql` and `pgAdmin III`. `pgAdmin III` has a graphical interface, we shall use this now and later use `psql`.

2.9 Scalar and Vector Aggregates

When a single value is returned from an `SQL` query it is called a *scalar*. For example, `MIN`, `MAX`, `SUM` are some functions that will return a single value. A `GROUP BY` clause will return several values, collectively these values are called *vector*.

2.10 `NULL` values

A datatype in SQL can have a NULL value, there can be three possibilities;

1. a value is inappropriate or not possible
2. a value is not known at the time other data items are available, the value is appropriate
3. a value that has not been entered in the database, it may be known and is appropriate

Conisder the following data items in the Patient table shown in figure 2.1

Patient Name	**Date of Birth**	**Date of Death**	**Phone**
Wut Wutuaz	21 Dec 1901	23 Sept 2001	NULL
Kef Kefaeq	23 Mar 1956	NULL	613-725-2981
Yoy Yoyoiz	NULL	NULL	NULL
Lum Lumuef	5 Apr 1946	NULL	NULL

Table 2.1: Patient Table

In the first case Patient `Wut` is deceased, a phone number for that person may not be appropriate. The second patient `Kef` is alive, she has all details except date of death. Patient `Yoy` does not have any details except her name, and `Lum` does not have Date of Death and Phone number.

An example of the DDL statement to create a table to store the above data is shown in figure 2.6. A `NULL` specification does not mean that all values have to be `NULL`, it means `NULL` values are allowed in the field. In the DDL statement it is mandatory to have `Patient Name`, this field is explicitly defined `NOT NULL`, without a name it is not possible to add a record for that patient. Stated differently, it is not possible to add a record for a patient with the fields Date of Birth, Date of Death and Phone.

A `NULL` cannot be replaced by a zero or a space.

Exercise: Study table 5.2 on page 62, planets Mercury and Venus have NULL values for Satellies, what can you infer?

```
CREATE TABLE Patient(
Name        VARCHAR( 25 ) NOT NULL,
DateOfBirth DATE          NULL,
DateOfDeath DATE          NULL,
Phone       VARCHAR( 25 ) NULL );
```

Figure 2.6: DDL statement to create Patient Table

2.11 SQL String Functions

String functions manipulate and examine character data types - `CHAR`, `VARCHAR` and `TEXT`. A few commonly used string functions are shown in this section.

Concatenation

Concatenation is a string operation that joins two strings. Each language has its own string concatenation techniques. In SQL there are two ways you can join two or more character strings together. The first method is to use the function `CONCAT`. Try the following statements (one at a time) and observe the results.

```
SELECT CONCAT('Algonquin', 'College');
SELECT CONCAT('University', ' ', 'of', ' ', ' 'Ottawa');
```

The second method is to use the `||` operator. Type in the following statements,

```
SELECT ('Canada' || 'Day' );
SELECT ('Canada' || ' ' || 'Day');
```

Misc String Functions

Table 2.2 lists several comparison operators. An example illustrates usage in the `world` database.

Operator	Description

`UPPER( )`	Convert a string to uppercase. The SQL statement below will display country names in uppercase.
	`SELECT UPPER( Name )`
	`FROM Country;`
	`SELECT Name FROM City WHERE CountryCode = UPPER( 'mex ');`

`LOWER( )`	Convert string to lower case.
	`SELECT LOWER( Name ) AS "City", District`
	`FROM City`
	`WHERE CountryCode = 'RUS'`
	`AND District = 'Tomsk';`

`SUBSTRING( string`
`< FROM int >`
`< FOR int > )`
	Extract a substring.
	To display the `CountryCode` and first four characters of cities in Ecuador
	`SELECT CountryCode, SUBSTRING( Name FROM 1 FOR 4 )`
	`FROM City`
	`WHERE CountryCode = 'ECU';`

Table 2.2: SQL String Manipulation Functions

2.12 SQL Comparison Operators

Table 2.3 lists several comparison operators. An example illustrates usage in the `world` database.

Operator	Description
`=`	equal to.
`< >` or `!=`	NOT equal to.
`<`	Less than.
`>`	Greater than.
`<=`	Less than or equal to.
`>=`	Greater than or equal.
`IN`	Matches a set of values.
`NOT IN`	Excludes values in a given set.
`BETWEEN`	Matches values in between two given boundary values - including end values.
`NOT BETWEEN`	All values except those in the given range.
`LIKE`	Matches a set of characters in a pattern. Refer section 2.13 for an explanation.
`NOT LIKE`	Does not match a pattern of characters.
`IS NULL`	Use when filtering `NULL` values. Note: `=` sign is not functional when filtering `NULL` values.
`IS NOT NULL`	Gives rows that have values in attributes, i.e. excludes `NULL` values.

Table 2.3: SQL Comparison Operators

2.13 Difference between `LIKE` and `=`

SQL Tutorial provides this definition of the `LIKE` operator

> The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

`LIKE` is a commonly used pattern matching operator
Aside: `LIKE` allows matching text with simple patterns that can include two wild cards `%` and `_`

- `%` matches zero or more characters
- `_` (underscore) matches a single character

The three queries listed below differentiates between `LIKE` and `=`

```
-- this code will work
SELECT * FROM city
WHERE CountryCode LIKE 'CHN%';

-- this code will not work
SELECT * FROM city
WHERE CountryCode = 'CHN%';

-- this code will work
SELECT * FROM city
WHERE CountryCode = 'CHN';
```

Aside: `-- this query does not return any rows, % is replaced by *`

```
SELECT * FROM city
WHERE countrycode LIKE 'CHN*';
```

2.14 SQL Logical Operators

Table 2.4 lists several comparison operators.

Operator	Description
`AND`	Will list rows that match both conditions.
`OR`	Will list rows that match any one condition.
`NOT`	Negates an operand.

Table 2.4: SQL Logical Operators

2.15 SQL Aggregate Functions

Table 2.5 lists aggregate functions in SQL. An example illustrates usage in the `world` database.

Function	Description
`COUNT( * )`	Counts the number of rows in a table.
`COUNT( attribute )`	Counts the number of rows in a table that have values in the attribute specified. It does not match `NULL` values.
`SUM`	Adds all numeric values of an attribute in all rows of the table.
`AVG`	Calculates the average of numeric values of an attribute of all rows in the table.
`MIN`	Finds the smallest value of an attribute from all rows.
`MAX`	Finds the largest value of an attribute from all rows in a table.

Table 2.5: SQL Aggregate Functions

2.16 Alias

An alias serves several needs in an `SQL` statement, it improves readability by assigning meaningful names in a query, especially when aggregate functions are used. For example, in the following query the `City` table is aliased as `Ci` and `Country` table as `Co`. The last clause is abbreviated as `Ci.CountryCode = Cn.Code` - a compacted form.

```
SELECT Ci.Name AS "City", District, Cn.Name AS "Country"
FROM City Ci, Country Cn
WHERE CountryCode = 'RUS'
AND District = 'Tomsk'
AND Ci.CountryCode = Cn.Code;
```

2.17 VIEW

Views are of two types - Dynamic View and Materialized View. A Dynamic view is also called virtual table or logical view. Each of the views, dynamic and materialized have an advantage.

Dynamic View The result set does not take hard disk space. Dynamic view occupies space in the DBMS as an object. Each time a dynamic view is invoked, the query (view) is run by referring the tables.

A complex, or cumbersome, query can be simplified by creating a dynamic view. When the view is invoked as a `SELECT` statement the query is run. Any change made to the tables will affect the result set each time the view is invoked, hence the name *dynamic* view. A dynamic view will always provide the most recent data.

Example: To create a dynamic view you would type

```
CREATE VIEW Canada_V AS
SELECT * FROM City
WHERE CountryCode = 'CAN';
```

A view is dropped by the following command line `DROP VIEW IF EXISTS Canada_V;`

The result set from the view created above can be observed using
`SELECT * FROM Canada_V;`

Achieving Physical Data Independence is an important advantage of a dynamic view. An appreciation of data independence will come after database design topics have been covered.

Materialized View is a persistent view. A data set is created from the tables and stored in the database. The result set takes storage space, unlike a dynamic view where the result set does not occupy hard disk space. Each time a materialized view is run the result set that is stored is referenced.

Example: To create a materialized view you would type

```
CREATE MATERIALIZED VIEW Canada_MV AS
SELECT * FROM City
WHERE CountryCode = 'CAN';
```

A materialized view is dropped by the following command line
`DROP MATERIALIZED VIEW IF EXISTS Canada_MV;`

The result set from the materialized view created above can be observed using `SELECT * FROM Canada_MV;`

At this stage both views give the same result. Observe the schema in `pgAdmin`, materialized view has a different icon. *Note:* Both `CREATE` and `DROP` are `DDL` statements, `SELECT` are `DML` statements.

The difference between both these views are observed using the example below.

Observe the results from two `SELECT` statements above, pay attention to the number of rows in each case, they should be the same.

Add a new city to the city table, for example.

```
INSERT INTO City( ID, Name, CountryCode, District, Population )
VALUES( 8000, 'Gananoque', 'CAN', 'Ontario', 50000 );
```

Now run each of the two VIEWs again.
First from the dynamic view
`SELECT * FROM Canada_V;`
and then from the materialized view
`SELECT * FROM Canada_MV;`

The dynamic view will show the updated result from the city table, the materialized view will not show the newly added row.

A materialized view can be refreshed using the following syntax:

```
REFRESH MATERIALIZED VIEW Canada_MV;
```

Now run each of the two VIEWs once again (separately), after refreshing the materialized view the result is recent and it now matches the most recent data in the table.

```
SELECT * FROM Canada_V;
SELECT * FROM Canada_MV;
```

Convince yourself how each of the two view differ.

As an additional exercise, delete the newly added row from the city table using:

```
DELETE FROM City WHERE ID = 8000;
```

Observe the result set from the two views

```
SELECT * FROM Canada_V;
SELECT * FROM Canada_MV;
```

Once again, observe that the materialized view does not show the updated result set. What would you do to update the materialized view?

A materialized view has its advantages. Often the most recent data is not required, for example data from the previous day is acceptable. Queries that are demanding on the computers resources, CPU and storage, are candidates for materialized views. Users will be able to access the data without overloading the system. Periodically the materialized view is refreshed to update the data. Usually, developers and end users will work on views instead of tables. A data administrator will create views from subsets of tables filtering certain rows, columns or both rows and columns, that have sensitive data. Users are granted permission to use views but not tables. This method allows access control.

Review Questions on VIEW

1. What are the two types of views?

2. Write two advantages of a view.

2.18 Data Types

Data Type is classification of data. Some considerations when choosing a datatype are: the *kind* of data, how the user/programmer intends to *use* the data, the *operations* that are needed to perform on the data, the *range* - i.e. maximum or minimum values that the data value could take. Postgres supports a variety of datatypes. This section briefly describes the common datatypes. The sample `DDL` statements are used to illustrate the explanation that follows.

```
-- Demonstrate and document DataTypes.
-- DROP TABLE IF EXISTS Student_T;

CREATE TABLE Student_T(
   ID          CHAR( 9 ),
   FirstName   VARCHAR( 20 ),
   LastName    VARYING CHARACTER( 20 ),
   DOB         DATE,
   CareerGoal  TEXT,
   Balance     DECIMAL( 10, 2 )
);
-- eof: DataType.sql
```

Figure 2.7: Datatype, an Illustration

Character Data Types

This category has three main types, `CHAR`, `VARCHAR` and `TEXT`.

`CHARACTER` and `CHARACTER VARYING` These are two basic character data types. They are shortened as `CHAR` and `VARCHAR`. The number within the brackets indicate the maximum length.

`CHAR( 9 )` will store a maximum of 9 characters. The storage space on the storage media required for `CHAR`, in this case will be 9 characters. A data item of less than 9 characters will still take up 9 characters of storage space.

`VARCHAR( 20 )` In both data types, `CHAR( n )` and `VARCHAR( n )`, storing more than the number of maximum characters will result in an error; unless the data is all spaces, it will truncate the number of spaces stored to *n*. Specifying `CHAR` without a length will default to one character. Not specifying the number of characters in `VARCHAR`, a very large string (upto 1GB) can be stored.

Aside: By explicitly casting `CHAR` or `VARCHAR` to `n` characters no error is raised. This topic is not covered in this course. Refer to the postgreSQL reference manual for details.

Numeric Data Types

This data type allows storage of numerical values - integer and floating point. Integer values can take 2, 4 or 8 bytes to store, floating point values can take 4 or 8 bytes.

`INTEGER` takes 4 bytes of storage space, the range of values are from `-2,147,483,648` to `+2,147,483,647`. *Aside:* The total number of unique values are `2,147,483,648 + 2,147,483,647 = 4,294,967,295`, including the zero value. This value is equal to $2^{32} - 1$, 32 bits correspond to 4 bytes.

`BIGINT` takes 8 bytes of storage, it stores values from -2^{63} to $+2^{63}$.

`DECIMAL` The storage required to store a decimal number is dependent on the definition. For example, `DECIMAL( 5, 2 )`, will store a dollar values from `$999.99` to `$-999.99`. Entering a value `99.999` will round it to `100.00`, similarly `-99.999` will round it to `-100.0`. To store the value of your home you may need `DECIMAL( 6, 0 )`, or `DECIMAL( 8, 0 )` after the stock options from Silicon Valley have materialized.

Storing Date, Date & Time

`DATE` **and** `DATETIME` The two most common data type for storing date are `DATE` and `TIMESTAMP`. `DATE` will takeup 4 bytes of storage, it can record dates from 4713 BC until 5874897 AD, for all business applications `DATE` is sufficient, it does not record time. `TIMESTAMP` takes 8 bytes of storage, it stores date *and* time. These two data types are not sufficient for storing very small time values, such as in physics or large durations for astronomical calculations. Postgres does have other datatypes that could possible handle these values.

Storing Binary and Character Objects

`BLOB` is an acronym for Binary Large Object; this data type used to store binary data for example - image, audio, video or executable files. Its use is not standard across DBMS's. `BLOB`'s may be used to store a few critical data items, for large applications such as social media sites using a BLOB data types will have data management concerns such as backup and recovery. In such cases other methods are used.
`CLOB` or Character Large Object is another data type similar to `BLOB`, but stores character data. Its maximum size is limited to 4GB, per record (compare to 2000 bytes for `CHAR`); this makes a SELECT clause slow and impractical. An alternative arrangement could be to store a reference to the data item in the `CLOB` field.

2.19 Review Questions

The SQL Environment

Answer True or False

1. A composite key is a primary key that consists of more than one attribute
   (a) True
   (b) False
2. A rule that states that each foreign key value must match a primary key value in the other relation is called the referential integrity constraint.
   (a) True
   (b) False

   For the relationship represented in figure 2.8, answer the following three questions

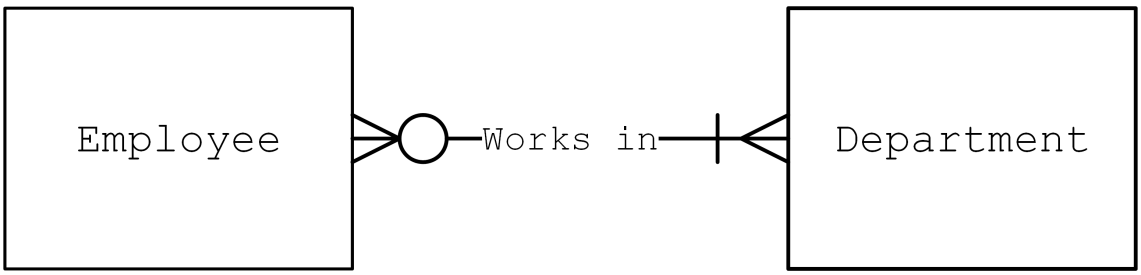


Figure 2.8: Employee-Department Relationship

3. A department can have more than one employee
   (a) True
   (b) False
4. It is possible that an employee does not belong to any department
   (a) True
   (b) False
5. It is possible that a department may not have any employees
   (a) True
   (b) False

Multiple Choice Questions.

Select the best answer. Each question has one correct answer.

1. A primary key
   (a) can be `NULL`
   (b) cannot be `NULL`
2. Referring to the `world` database, a code segment in `DDL` to create the `City` table is shown below. What does the term `SERIAL` indicate?

```
CREATE TABLE City (
ID SERIAL PRIMARY KEY,
... ...
```

   (a) it has no meaning, `SERIAL` is an outdated syntax
   (b) city ID is a surrogte key and values are added automatically in serial order
   (c) it is a special case, primary key can be `NULL`

3. Referring to the world database - Which SQL statement will remove the column `IsOfficial` from table `CountryLanguage`

   (a) `ALTER TABLE CountryLanguage.IsOfficial drop column;`
   (b) `ALTER TABLE CountryLanguage delete column IsOfficial;`
   (c) `ALTER TABLE CountryLanguage drop column IsOfficial;`
   (d) `ALTER TABLE CountryLanguage drop field countryLanguage.IsOfficial;`
   (e) `ALTER TABLE CountryLanguage.IsOfficial delete column;`

4. Identify the command that adds a column to the `CITY` table in the world database. The column is titled `area` and stores the area in square kilometers rounded to the nearest kilometer, and able to easily perform simple arithmetic calculations on the area.

   (a) `MODIFY TABLE CITY ADD area INT( 6 );`
   (b) `MODIFY TABLE CITY ADD area VARCHAR( 10 );`
   (c) `ALTER TABLE CITY ADD area INT( 6 );`
   (d) `ALTER TABLE CITY ADD area CHAR( 10 );`
   (e) `ALTER TABLE CITY ADD area VARCHAR( 10 );`

5. The _____ is the structure that contains descriptions of objects such as tables and views created by users.

   (a) SQL
   (b) schema
   (c) master view

6. The SQL command _____ defines a logical table (virtual table) from one or more tables or views.

   (a) create relationship
   (b) alter table
   (c) create view
   (d) create table
   (e) alter view

7. A `CREATE` command adds a table. Identify the command that will remove the table.

   (a) `DELETE`
   (b) `DROP`
   (c) `TRUNCATE`
   (d) `UNPACK`

8. The first in a series of steps to follow when creating a table is to:

   (a) identify columns that must be null
   (b) create an index
   (c) identify columns that must be unique
   (d) identify each attribute and its characteristics

9. Identify the SQL command that adds or deletes a column from a table.

   (a) `CREATE VIEW`
   (b) `CREATE TABLE`
   (c) `ADD COLUMN` or `DELETE COLUMN`
   (d) `ALTER TABLE`
   (e) `MODIFY TABLE`

10. Identify the statement that inserts a row in the city table with city Oshawa in Ontario.
    (a) `INSERT CITY VALUES( '6002', 'Ontario', 'CAN', 'Oshawa', 140000 );`
    (b) `ADD INTO CITY( CountryCode, District, ID, Name, Population )`
    `VALUES ( 'CAN', 'Ontario', 6002, 'Oshawa', 140000 );`
    (c) `MODIFY INTO CITY( CountryCode, District, ID, Name, Population )`
    `VALUES( 'CAN', 'Ontario', 6002, 'Oshawa', 140000 );`
    (d) `INSERT INTO CITY VALUES( '6002', 'Oshawa', 'CAN', 'Ontario', 140000 );`
    (e) `INSERT INTO CITY( CountryCode, District, ID, Name, Population )`
    `VALUES( 'CAN', 'Ontario', 6002, 'Oshawa', 140000 );`
11. The command
    `INSERT INTO CITY VALUES( 'British Columbia', 'Kelowna' );` is used to add city Kelowna in the Name column, and British Columbia in the district column. The SQL command will not work because
    (a) The keyword table is missing
    (b) The correct command is modify
    (c) The field names must be specified if inserting some of the fields
    (d) Possibly because Kelowna already exists in the table
12. What will the following statement do
    `DELETE from city where countrycode = 'NDL';`
    (a) Deletes all countries which have code beginning with `NDL`
    (b) Deletes all records (rows) from the city table which has countrycode as `NDL`
    (c) Deletes all columns (attributes) from the city table which has countrycode as `NDL`
    (d) Delete all tables which begin with `NDL`
13. Identify the statement that will change the population of city Zaanstad to 136621 in the CITY table
    (a) `UPDATE CITY SET population = '136621' WHERE Name = 'Zaanstad';`
    (b) `MODIFY CITY SET population = 136621 WHERE Name = 'Zaanstad';`
    (c) `MODIFY COUNTRY SET population = 136621 WHERE Name = 'Zaanstad';`
    (d) `UPDATE CITY SET population = 136621 WHERE Name = 'Zaanstad';`
    (e) `UPDATE COUNTRY SET population = 136621 WHERE Name = 'Zaanstad';`
14. In an SQL statement, identify the clause that will specify conditions for row selection:
    (a) `FROM`
    (b) `SELECT`
    (c) `WHERE`
    (d) `GROUP BY`
    (e) `ORDER BY`
15. In an SQL statement, identify the clause that will select tables:
    (a) `FROM`
    (b) `ORDER BY`
    (c) `WHERE`
16. Identify the qualifier that will not display duplicate rows in a SQL query.
    (a) `SPECIFIC`
    (b) `NO DUPLICATE`
    (c) `ALTER`
    (d) `DISTINCT`
    (e) `UNIQUE`
17. Data combined from several observations is called
    (a) summary data
    (b) aggregate
    (c) score

(d) standard deviation
(e) average

18. A single value returned from an SQL query that includes an aggregate function

(a) schema
(b) dynamic view
(c) base table
(d) scalar
(e) vector

19. Multiple values returned from an SQL query that includes an aggregate function

(a) schema
(b) dynamic view
(c) base table
(d) scalar
(e) vector

20. Which SQL clause will sort the rows in ascending or descending order?

(a) `GROUP BY`
(b) `HAVING`
(c) `ORDER BY`

21. An attribute or a combination of attributes that uniquely identifies each row in a relation is

(a) Foreign Key
(b) Prime Key
(c) Composite Attribute
(d) Multivalued Attribute

22. An attribute in a relation that serves as the primary key of another relation in the same database.

(a) Foreign Key
(b) Prime Key
(c) Composite Attribute
(d) Multivalued Attribute

Operator Precedence

23. Identify the correct processing order of boolean operators

(a) `NOT` then `OR` then `AND`
(b) `NOT` then `AND` then `OR`
(c) `AND` then `OR` then `NOT`
(d) `AND` then `NOT` then `OR`
(e) `OR` then `AND` then `NOT`

24. What will be result from the following SQL statement?

```
SELECT Part
FROM Inventory
WHERE Part = 'Bolt' OR Part = 'Nut' AND Material = 'Brass' OR Material = 'Steel'
```

(a) Nuts or Bolts made from Brass or Steel
(b) Nuts made from Brass, all bolts and all steel parts
(c) Bolts made from Brass and Nuts made from Steel
(d) Either nuts or bolts, made from either brass or steel
(e) Bolts made from Steel or Nuts made from Brass

Written Questions

1. Phone numbers are represented by numerals but are defined as characters in a database. Give reasons to support this statement.

2. Write three advantages of using a standard language such as SQL.

3. Briefly explain Data Definition Language. Give examples of commands that qualify as Data Definition Language.

2.20 GROUP BY

`GROUP BY` is used with aggregate functions, it provides a summary of rows. Examples of aggregate functions are `AVG`, `SUM` and `COUNT`.

Use `world` database to test the queries in this section.

```
SELECT SUM( Population )
FROM City;
```

Figure 2.9: Query to add population of all cities

The above query will give us one number which is the total of all cities in the City table. If we want to breakup this sum by each country, the query needs to be extended.

The following SQL statement provides the sum of the number of people living in cities in each of the countries:

```
SELECT CountryCode, SUM( Population )
FROM City
GROUP BY CountryCode;
```

Figure 2.10: Using `GROUP BY` clause

Run the query in figure 2.10 and observe the results.

Now, remove the GROUP BY clause and run the query again. Does the query work?

```
SELECT CountryCode, SUM( Population )
FROM City;
```

Figure 2.11: Using an Aggregate function without a GROUP BY clause

The query in figure 2.11 will give an error, CountryCode must be part of the GROUP BY clause. *Aside:* Versions of MySQL will run the query without error but will give an incorrect result. As an SQL user, always anticipate the expected result and then confirm with the result provided by the query.

2.20.1 GROUP BY and HAVING

HAVING clause qualifies groups. Expanding the query in figure 2.10, to limit countries with the sum of city populations less than 200000.

```
SELECT CountryCode, SUM( Population ) AS 'Sum Population'
FROM city
GROUP BY CountryCode
HAVING SUM( Population ) < 200000;
```

Figure 2.12: GROUP BY qualified by HAVING

In figure 2.12 the HAVING clause qualifies the countries. Observe the same aggregate SUM(Population) in the expression list of the SELECT caluse and HAVING clause. WHERE clause does not allow aggregates, it qualifies rows in a table.

Caution: The following query although syntactically correct gives incorrect results. In the processing order in figure 2.2 on page 13 HAVING clause is processed before the SELECT clause. An alias cannot be used in the HAVING clause.

```
SELECT CountryCode, SUM( population ) as 'Sum Population'
FROM City
GROUP BY CountryCode
HAVING 'Sum Population' < 200000;
```

Figure 2.13: Syntactically correct query with incorrect results

Code in Figure 2.14 will list Countries with more than five cities.
Aside: Notice the alias after AS has the the name is double quotes

Exercise The code does not list the name of the country, just the CountryCode - modify the query to list the name of the country with CountryCode.

```
SELECT CountryCode, COUNT( CountryCode )  AS "Number of Cities"
FROM City
GROUP BY CountryCode
HAVING COUNT( CountryCode ) >= 5;
```

Figure 2.14: Using GROUP BY to list Countries With More Than 5 Cities

Exercise What is the result of the code in figure 2.15

```
SELECT CountryCode, SUM( Population ) AS SUM
FROM City
GROUP BY CountryCode
ORDER BY SUM DESC;
```

Figure 2.15: GROUP BY - Exercise

2.20.2 Examples of Some Invalid Queries

1. Aggregate functions cannot be used in a `WHERE` clause. Example,
   ```
   SELECT City, Population
   FROM City
   WHERE Population = MAX( Population );
   ```

2. Subqueries cannot be used in aggregate functions. Example,
   ```
   SELECT AVG( SELECT Population FROM Country )...
   ```

3. Aggregate functions cannot be nested. Subqueries must be used to get the desired result. *Exception:* Oracle allows nested aggregate functions.
   Example,
   ```
   SELECT AVG( MAX( Population) )
   ```

2.20.3 Review Questions - `GROUP BY`

1. Which of the following can produce scalar and vector aggregates?
   (a) `ORDER BY`
   (b) `HAVING`
   (c) `GROUP BY`
2. An aggregate function, for example `MAX`, `SUM`, can be used in a `WHERE` clause.
   (a) True
   (b) False
3. An aggregate function, and a non aggregate function can be used in a `SELECT` clause only if a `GROUP BY` clause is used for the scalar value.
   (a) True
   (b) False
4. Aggregate functions cannot be nested, for example, `SELECT AVG( SUM( population ) )` is an invalid statement in SQL.
   (a) True
   (b) False

2.21 Summary

SQL, though not a perfect query language has its benefits. It is a declarative language with procedural elements;

you do not have to declare variables before using them. Its widespread use in industry makes it an important skill to have.

3

JOIN

3.1 Introduction

This chapter addresses multi table queries, querying multiple tables include `joins` and `sub-queries`. Two other topics critical to web based development are embedded SQL and dynamic SQL. Sub-query is a versatile query technique in SQL, it is used to return values to another query.

3.2 Objective

- Build syntax for `JOIN`s - equijoin, natural join, outer join, inner join, self join
- Write sub-queries. Differentiate and compare sub-query with join
- Build equivalent queries using `JOIN` and sub-query
- Compare correlated and non-correlated sub-queries
- Combine queries using `UNION`
- `CREATE` and `DROP` Indexes
- Use conditional expressions
- Recognize the need for Transaction Management
- Determine the use and location of triggers and stored procedures. Differentiate stored procedures and triggers
- Differentiate procedures and functions
- Determine the need for embedded SQL and dynamic SQL in web based applications

3.3 Working with more than one table

Table 3.1 lists types of joins with a brief explanation. Note, `CROSS JOIN` and `UNION` are not join operations.

3.4 JOIN

Definition: A `JOIN` clause combines one or more tables based on a common data value. The tables are from the same database.

Data is stored in individual tables; it is the *relationship* between the tables that make the data meaningful; `JOIN`s perform that meaningful relationship. [4, Fehily, Page 193]. `SQL`'s strength is in `JOIN` operations.

Join	**Result**
NATURAL JOIN	All requested results, values of common columns returned only once. Duplicate column eliminated. If there are no common column names, `NATURAL JOIN` results into a `CARTESIAN PRODUCT` (`CROSS JOIN`)
INNER JOIN	Requires each record in two tables to have matching records. Used in many situations. May not be suitable in some conditions.
EQUI JOIN	Uses only field values in tables that are equal. Returns all requested results including values of common columns.
OUTER JOIN	Returns all values from one table even if match not found.
LEFT OUTER JOIN	Joins rows based on matched values. The result includes unmatched rows from the table on the **left** of the `JOIN` clause.
RIGHT OUTER JOIN	Joins rows based on matched values. The result includes unmatched rows from the table on the **right** of the `JOIN` clause.
FULL OUTER JOIN	Joins rows based on matched values and returns rows from both tables. Not supported in MySQL.
SELF JOIN	Joins the table to itself.

Table 3.1: Joins

Operation	**Result**
CROSS JOIN	Cartesian product of tables. Combines each row from the first table with each row from the second table.
UNION	Returns a table that includes all data from each table. Both tables (or views) must have the same number of columns and the data types of both columns must be same. Strictly, not a join.

Table 3.2: Other Operations

NATURAL JOIN

Result of a `NATURAL JOIN` is a set of records from table 1 and table 2 that are equal in their common attribute.

A NATURAL JOIN is one of the eight operators. The following example is adopted from back inside cover of [1]. Given minimal metadata, perform a `NATURAL JOIN`.

a1	b1
a2	b1
a3	b2

Table 3.3: Table 1

b1	c1
b2	c2
b3	c3

Table 3.4: Table 2

Exercise - Natural Join

Perform a `NATURAL JOIN` on the following tables.

p1	x2
p2	x2
p3	x4

Table 3.5: Table A

x2	q1
x3	q2
x4	q3

Table 3.6: Table B

Using `JOIN`'s

Consider the following three tables `Team`, `Player` and `PlayerTeam`. Table `Team` has five Teams, `TeamID` is the prime-key. Table `Player` has six players. Players are assigned to teams, this is shown in table `PlayerTeam`.

TeamID	**Team**
21	Woqag
22	Zumey
23	Gavop
24	Turoz
25	Nibeg

Table 3.7: Team

PlayerID	**PlayerName**
445	Bix
446	Lay
447	Vow
448	Cox
449	Sal
450	Kar

Table 3.8: Player

PlayerID	**TeamID**
445	21
446	23
447	23
448	24
449	24

Table 3.9: PlayerTeam

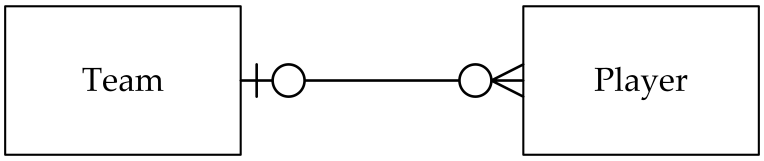


Figure 3.1: Logical ERD, Player Team

Player `Kar` has not been assigned a Team; Team `25,Nibeg` and Team `22,Zumey` has no players assigned.

3.4.1 Exercise

Use `PlayerID` and `TeamID` from Table 3.9 to write, in tabular form, Player Name and Team Name.

INNER JOIN

`INNER JOIN` is the most common type of join - although it is not suitable in all situations. It requires each record in two tables to have matching records. Here is an `INNER JOIN` that gives the lists Players and Team assigned to the player. The SQL statement in figure 3.2 joins three tables:

```
SELECT PlayerName, TeamName
FROM Player
INNER JOIN
PlayerTeam USING( PlayerID )
INNER JOIN
Team USING( TeamID );
```

Figure 3.2: Inner Join

`INNER JOIN` has predictable results when referential integrity is enforced. Referential Integrity needs to be preserved if the results of `INNER JOIN` have to match the expected results. In sample tables, Player **Kar** does not appear in the result - he/she is not associated with a team. Also, team `25, Nibeg`, and team `22, Zumey` does not appear in the results.

An `INNER JOIN` is suited for tables that do not have `NULL` values. Data with `NULL` values are omitted without error or warnings. `NULL` values in one table **do not** match any values in another table, not even `NULL` values.

LEFT OUTER JOIN

`LEFT JOIN` joins two tables; the result matches attributes from both tables *and* it includes unmatched rows from the table on the **left** of the `JOIN` clause.

```
-- LEFT OUTER JOIN
-- Showing all players and their teams.
-- Showing player Kar does not belong to any team
SELECT Player.PlayerID, PlayerName, PlayerTeam.TeamID
FROM Player
LEFT JOIN PlayerTeam
ON PlayerTeam.PlayerID = Player.PlayerID;
```

Figure 3.3: Left Outer Join

```
-- LEFT OUTER JOIN
-- Player Kar does not belong to any team
SELECT Player.PlayerID, PlayerName, PlayerTeam.TeamID
FROM Player
LEFT JOIN PlayerTeam
ON PlayerTeam.PlayerID = Player.PlayerID WHERE PlayerTeam.PlayerID IS NULL;
```

Figure 3.4: Left Outer Join using `IS NULL`

```
-- LEFT OUTER JOIN
-- Showing team Zumey and team Nibeg do not have any players
SELECT Team.TeamID, TeamName, PlayerTeam.PlayerID
FROM Team
LEFT JOIN PlayerTeam
ON PlayerTeam.TeamID = Team.TeamID;
```

Figure 3.5: Left Outer Join

```
-- NOT IN clause
-- Note the use of a sub-query -- Showing team Zumey and team Nibeg do not have any players
SELECT Player.PlayerID, PlayerName
FROM Player
WHERE Player.PlayerID
NOT IN( SELECT PlayerID FROM PlayerTeam);
```

Figure 3.6: `NOT IN` clause

Aside: Equivalent SQL statement using `WHERE` clause to get the same result from query in figure 3.5. Type and run this query, you will notice the order of rows is different. Using and `ORDER BY` clause will not work directly, SQL needs both queries in the `UNION` clause to be ordered. This solution is a hack, it forces a `NULL` in the second statement.

```
SELECT Team.TeamID, TeamName, PlayerTeam.PlayerID
FROM
Team, PlayerTeam
WHERE Team.TeamID = PlayerTeam.TeamID
UNION
SELECT TeamID, TeamName, NULL
FROM Team
WHERE
TeamID NOT IN( SELECT TeamID FROM PlayerTeam );
```

RIGHT OUTER JOIN

`RIGHT JOIN` joins two tables; the result matches attributes from both tables *and* it includes unmatched rows from the table that is on the **right** of the `JOIN` clause.

```
-- RIGHT OUTER JOIN
-- Showing team Zumey and team Nibeg do not belong to any players
SELECT Team.TeamID, TeamName, PlayerTeam.PlayerID
FROM PlayerTeam RIGHT JOIN
Team ON PlayerTeam.TeamID = Team.TeamID;
```

Figure 3.7: Right Outer Join

Tips on using `OUTER JOIN`

1. Equate data items that have common values. For example, referring to 3.7 using the phrase
`ON PlayerTeam.TeamID = Team.TeamName`
will not yield any results (or incorrect results.) `TeamID` and `TeamName` are different data items. *Note:* The data items must be same, not the field names. `City.CountryCode = Country.Code` will give results - `CountryCode` in the `city` table has matching values in the `country` table in the field name `Code`.

2. Make sure the list of attributes in the `SELECT` clause have meaningful attributes. The `JOIN` will give you the results but if the field names are not correctly chosen you will get invalid rows.

FULL OUTER JOIN

Joins rows based on matched values and returns rows from both tables. Not supported in MySQL; not often used. Same results are obtained by using `RIGHT OUTER JOIN`, `UNION` with `LEFT OUTER JOIN`.

SELF JOIN

A table joined with itself is a `SELF JOIN`. It happens in a unary relationship, when a table with a foreign key references the primary key in the same table. Think of a join on two tables which are same; each row of one table is combined with each row of the other table. There is no explicit statement for a `SELF JOIN`. Examples of `SELF JOIN`: Consider an employee table, with `EmployeeID`, `Name` and `ManagerID`. Most employees will have a manager whose ID will be associated with the employee record.

A drug will have contraindications with another drug. `DrugID`, `Name` and `ContradicationID` will be in the same record.

Exercise: Run the script `SELF JOIN Demo.sql` and `SELF JOIN Query.sql`

CROSS JOIN

A cross join does not apply any predicate or filter, it has limited practical use in data processing. Be careful when using a cross join, the result is often not what you expect. Mathematically, a `CROSS JOIN` is a `product`, (specifically a cartesian product) one among the *original eight* operators. A `NATURAL JOIN`, discussed later, is a join operator. Always examine the result of a join operation and verify with your expected outcome. An incorrectly specified `WHERE` clause results in a cross join

Aside: A predicate is an operator, or a function, that returns a `TRUE` or `FALSE`.

A cross join can be run in two ways:
`SELECT * FROM <tableA> CROSS JOIN <tableB>;`
or simply
`SELECT * FROM <tableA>, <tableB>;`

`SELECT * FROM Team, Player;` results in 30 rows. In this case it is not a useful result.

Question: Think of a use of a cartesian product, i.e. `CROSS JOIN`

Exercise: Run the script `deck.sql`.
Aside: Note the method of inserting rows using a single `INSERT` statement

Final Note

In MySQL, `JOIN` and `INNER JOIN` are syntactic equivalents, they can replace each other. In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON clause`

1. A join operation:
   (a) is used to combine indexing operations
   (b) joins two tables with a common attribute to form a single table or view, the common attribute must be a prime key in both tables
   (c) joins two tables with a common attribute to be combined into a single table or view
   (d) joins two disparate tables to be combined into a single table or view
2. A join operation is performed on two tables. The common field that is used for the join operation has a few `NULL` values in each of the two tables. The join operation will:
   (a) match `NULL` values from the first table but not from the second table
   (b) match `NULL` values from the second table but not from the first table
   (c) match `NULL` values from both tables, since it is a join operation
   (d) not match `NULL` records from any of the two tables
3. Identify the clause that is used to combine output from multiple queries into a single result table.
   (a) `COLLATE`
   (b) `INTERSECT`
   (c) `DIVIDE`
   (d) `UNION`
   (e) `SELF JOIN`

4. `SELECT * FROM Student, Course;` will result in a
   (a) `NATURAL JOIN`
   (b) `SELF JOIN`
   (c) `OUTER JOIN`
   (d) `CROSS JOIN`
   (e) `INNER JOIN`
5. A join in which rows that do not have matching values in common columns are still included in the result table is called a/an:
   (a) outer join
   (b) union join
   (c) equi-join
   (d) natural join
   (e) inner join
6. An operation to join a table to itself is called a/an:
   (a) self join
   (b) inner join
   (c) natural join
   (d) outer join
   (e) equi join
7. An alternative term used for `CROSS JOIN` is
   (a) `NATURAL JOIN`
   (b) `SELF JOIN`
   (c) `OUTER JOIN`
   (d) `CARTESIAN PRODUCT`
   (e) `INNER JOIN`

1. Is the result of the `LEFT OUTER JOIN` the same as the table on the *left* of the `JOIN` clause? Justify your answer.

2. Perform a `NATURAL JOIN` on the tables below.

omlette	salt
omlette	pepper
sandwich	ketchup
poutine	vinegar
poutine	salt

Table 3.10: Food A

salt	beans
ketchup	burger
mustard	burger
sugar	cake

Table 3.11: Food B

3. Perform a `NATURAL JOIN` on the tables below.

omlette	salt
omlette	pepper
sandwich	ketchup
poutine	`NULL`
poutine	salt

Table 3.12: Food C

salt	beans
ketchup	burger
`NULL`	burger
sugar	cake

Table 3.13: Food D

3.5 Additional Example on JOIN Operations

Abstract This exercise illustrates JOIN operations between two tables, music players and the instruments they play in Table 3.14 and the Instruments and the category in table 3.15. Various JOIN operations are shown below. In the data shown below, a player plays only one instrument.

Player	Instrument
Glen Gould	Piano
Young Joo Chang	Violin
Guan Pinglu	Guqin
Hari P.C.	Bansuri
Seamus Egan	Bodran
Angela Hewitt	Piano
Alice Giles	Harp
Elina Karokhina	Balalaika

Table 3.14: Player_Instrument_T

Instrument	Category
Piano	String-Hybrid
Violin	String
Erhu	String
Bansuri	Wind
Guqin	String
Bagpipe	Wind
Bodhran	Percussion
Harp	String
Balalaika	String

Table 3.15: Instrument_T

3.5.1 DDL & DML statements

```
-- Filename: Player-Instrument-DDL-DML.sql
-- Author: S I'aret
-- Created: 3 Feb 2019
-- Note: This is a shorter way to insert data using a single INSERT statement
-- Caution: It is harder to debug and detect errors using this method.
-- It can be used for very small tables, such as this
-- Do not use this method for your assginments.
-- The syntax is terse, hence its usage here.

DROP TABLE IF EXISTS Player_Instrument_T;
DROP TABLE IF EXISTS Instrument_T;

CREATE TABLE IF NOT EXISTS Player_Instrument_T(
   Player       VARCHAR( 25 ),
   Instrument   VARCHAR( 20 )
);


CREATE TABLE IF NOT EXISTS Instrument_T(
   Instrument   VARCHAR( 20 ),
   Category     VARCHAR( 15 )
);
```

```
INSERT INTO Player_Instrument_T( Player, Instrument ) VALUES
( 'Glenn Gould', 'Piano' ), ( 'Angela Hewitt', 'Piano' ),
( 'Young Joo Chang', 'Violin' ), ( 'Guan Pinglu', 'Guqin' ),
( 'Hari P. C.', 'Bansuri' ), ( 'Seamus Egan', 'Bodhran' ),
( 'Alice Giles', 'Harp' ), ( 'Elina Karokhina', 'Balalaika' );


INSERT INTO Instrument_T( Instrument, Category ) VALUES
( 'Piano', 'String-Hybrid' ), ( 'Violin', 'String' ),
( 'Erhu', 'String' ), ( 'Guqin', 'String' ),
( 'Bagpipe', 'Wind' ), ( 'Bodhran', 'Percussion' ),
( 'Bansuri', 'Wind' ), ( 'Harp', 'String' ),
( 'Balaika', 'String' );


-- eof: Player-Instrument-DDL-DML.sql
```

3.5.2 Queries

```
SELECT * FROM Player_Instrument_T;
SELECT * FROM Instrument_T;

-- All Players with their instruments.
-- JOIN, USING
-- All three attributes are selected from their tables without ambiguity
SELECT Player, Instrument, Category
FROM Player_Instrument_T
JOIN Instrument_T USING( Instrument );


-- All Players with their instruments.
-- JOIN, ON
-- Column Instrument must be explicitly defined as Player_Instrument_T.Instrument
-- result is the same as JOIN, USING
SELECT Player, Player_Instrument_T.Instrument, Category
FROM Player_Instrument_T
JOIN Instrument_T ON Player_Instrument_T.Instrument = Instrument_T.Instrument;


-- All Players with their instruments.
-- NATURAL JOIN
SELECT Player, Instrument, Category
FROM Player_Instrument_T
NATURAL JOIN Instrument_T;


-- All Players with their instruments.
-- INNER JOIN
SELECT Player, Instrument, Category
FROM Player_Instrument_T
INNER JOIN Instrument_T USING( Instrument );
```

```
-- implicit JOIN
-- This query does not use the JOIN keyword
-- It gives the same result with a WHERE keyword
SELECT Instrument_T.Instrument, Player, Category
FROM Instrument_T, Player_Instrument_T
WHERE Instrument_T.Instrument = Player_Instrument_T.Instrument;


-- Instruments with no players
-- Query using a LEFT JOIN
SELECT Player, Instrument, Category
FROM Instrument_T
LEFT JOIN Player_Instrument_T USING( Instrument )
WHERE Player IS NULL;


-- unmatched values in Instrument_T with Player_Instrument_T
-- Instruments without Players
SELECT Instrument_T.Instrument, Category
FROM Instrument_T
WHERE Instrument_T.Instrument NOT IN( SELECT Instrument FROM Player_Instrument_T );


-- Instruments with no players, and players with an instrument
SELECT Player, Instrument, Category
FROM Instrument_T
LEFT JOIN Player_Instrument_T USING( Instrument );


SELECT Player, Instrument, Category
FROM Instrument_T
FULL OUTER JOIN Player_Instrument_T USING( Instrument )
WHERE Player IS NULL;
```

3.6 Review Questions

Advanced SQL

1. A type of query that is placed within a `WHERE` or `HAVING` clause of another query is called a/an:
   (a) PL/SQL
   (b) subquery
   (c) embedded SQL
   (d) dynamic SQL
   (e) trigger
2. Identify the clause that takes a value of `TRUE` if a subquery returns one or more rows in an intermediate results table.
   (a) `IN`
   (b) `HAVING`
   (c) `EXTENTS`
   (d) `EXISTS`
   (e) `WHERE`
3. Identifying specific attributes in the `SELECT` clause, instead of using `SELECT *` will help reduce network traffic
   (a) True
   (b) False
4. Which one of the two SQL statements will filter countries with the letter **C** occurring only in the middle of the three letter `CountryCode`
   (a)
   ```
   SELECT countrycode, population
   FROM city
   WHERE countrycode like '%C%';
   ```
   (b)
   ```
   SELECT countrycode, population
   FROM city
   WHERE countrycode like '_C_';
   ```
5. SQL statements in a program written in another language such as C or Java are called
   (a) dynamic SQL
   (b) embedded SQL
   (c) sub-query
   (d) join
   (e) union
6. A named set of SQL statements that are executed when a data modification occurs are called:
   (a) Sub-queries
   (b) Trapdoors
   (c) Stored procedures
   (d) Triggers
   (e) PL/SQL
7. Identify the commands that need to be called explicitly, i.e. they cannot run automatically when a transaction has taken place
   (a) Sub-queries
   (b) Trapdoors
   (c) Stored procedures
   (d) Triggers
   (e) PL/SQL
8. Which one of the following returns values and take input parameters
   (a) procedures
   (b) functions

9. Embedded SQL statements can create a flexible and accessible interface for the user
   (a) True
   (b) False
10. Embedded SQL statements help enforce security by granting permission to required applications instead of granting permission to users
   (a) True
   (b) False
11. SQL statements are built by DBMS at the time a user or procedure requests data from a DBMS or a transaction is performed. The name given to such a concept is
   (a) Dynamic view
   (b) Material view
   (c) Dynamic SQL
   (d) Triggers
   (e) PL/SQL

3.7 Summary

Several SQL implementations do not differential between `NATURAL JOIN`, `INNER JOIN` and `EQUI JOIN`. A Join operation can be performed using the `WHERE` clause giving the same result set in most cases, in the industry the explicit `JOIN` operator is preferred; the `JOIN` operator make the results predictable with the data has `NULL` values.

4

SQL

4.1 Sub-query

A query nested inside another query is known as a `subquery`. The nested query executes first, its result is equated to the *outer* query.

Figure 4.1 illustrates a subquery. It lists cities in Canada whose population is higher than the average of all cities in Canada. The inner most query
`( SELECT AVG( population ) FROM City )`
is run first, its result is then compared to the `WHERE` condition in the outer query. Type and run this query in the world database. Change the greater than sign to a less than sign, observe the result. A query illustrated in figure 4.1 is also called *non corelated subquery*, it is because the inner query is independent from the outer query. It is run only once, its results are then passed on to the outer query.

```
SELECT ID, Name, Population
FROM City
WHERE population > ( SELECT AVG( population ) FROM City )
AND CountryCode = 'CAN';
```

Figure 4.1: Sub Query

A `correlated` subquery, also known as a synchronized subquery, uses values from the outer query. The subquery is evaluated once for each row processed by the outer query. Figure 4.2 and figure 4.3 are examples of correlated subqueries. In the first example the query lists all cities which have their population less than all cities in that country. The subquery (i.e. inner query) is evaluated each time the outer query is evaluated. This is not an efficient way to get the results. *Aside:* Observe the use of an alias in the table name. Recall that an alias is not permitted in the `WHERE` clause.

In the second example the query is part of the `SELECT` clause. The using the `ROUND` function the result is less cluttered. For example,
`( SELECT ROUND( ( AVG( population ) ), 0 )`

```
SELECT ID, Name, CountryCode, population AS "City Population"
FROM City AS "Avg City Population"
WHERE Population < ( SELECT AVG( population )
FROM City
WHERE CountryCode = "Avg City Population".CountryCode );
```

Figure 4.2: Correlated Sub Query

```
SELECT CountryCode, Name, Population,
( SELECT ( AVG( population ) ) AS "Country Average"
FROM City WHERE CountryCode = City_T.CountryCode )
FROM City AS City_T;
```

Figure 4.3: Correlated Sub Query. Query in the SELECT Clause

1. In a non correlated sub query which query runs first
   (a) inner query
   (b) outer query
2. A type of subquery where processing the inner query depends on data from the outer query.
   (a) correlated
   (b) non-correlated
   (c) inner
   (d) outer
   (e) embedded
3. A subquery that is executed once for the entire outer query is
   (a) embedded
   (b) correlated
   (c) non-correlated
   (d) outer
   (e) dynamic
4. Which one of the sub-queries does not depend on data from the outer query
   (a) inner
   (b) embedded
   (c) correlated
   (d) non-correlated
   (e) outer

4.2 Learning Activities

1. Use the `world` database to generate correlated subqueries. Identify the *inner* and *outer* queries

4.3 Transaction Management

A transaction is a unit of work that changes the state of a database. Examples of transactions are `INSERT`, `UPDATE` and `DELETE` statements, among others. A `SELECT` statement does not alter the database, it is not a transaction. Often several SQL statements need to run as a single unit. For example, if we had one record insertion in the `Invoice_T` table and 2 records inserted in the `Invoice_Line_T` table, a total of 3 statements in two tables as shown in Figure 4.4.

```
INSERT INTO Invoice_T( Invoice_Number, Cust_Id, Invoice_Date )
VALUES( 'I23008', 'C006', '2011-12-15');

INSERT INTO Invoice_Line_T( Invoice_Number, Invoice_Line, Prod_Code, Line_Unit, Line_Price )
VALUES( 'I23008', 1, 'P2016', 3, 689.00 );
INSERT INTO Invoice_Line_T( Invoice_Number, Invoice_Line, Prod_Code, Line_Unit, Line_Price )
VALUES( 'I23008', 2, 'P2017', 3, 35.99 );
```

Figure 4.4: Transactions in Inventory Database

These three statements must be completed together or not at all. If only the first or the first two statements ran successfully and the remaining statements were not run, the data would be inaccurate and even inconsistent.

Transaction Management allows to run all statements together or none at all. To do this the statements are wrapped around `BEGIN TRANSACTION` and `END TRANSACTION`, if there is an error the statements are nullified using the `ROLLBACK` command.

The exercise below illustrates how `ROLLBACK` will affect your commands.

1. start `psql` There are several ways to do this.
2. List all databases on server: `\l`
3. The command to connect to a database is: `\c <database>`. Connect to Inventory: `\c Inventory`
4. List all tables in the database: `\dt`
5. By default AUTOCOMMIT is ON, i.e all commands will automatically will be permanent.
   Turn AUTOCOMMIT OFF, type in. `\set AUTOCOMMIT off`. Now commands can be reversed.
6. Display records in `Invoice_Line_T;`
   `SELECT * FROM Invoice_Line_T;`
7. Delete records from table. `DELETE FROM Invoice_Line_T;`
8. Check records, `SELECT * FROM Invoice_Line_T;`
   Do you see a listing?
9. Type in: `ROLLBACK;` Your commands are now reversed.
10. `SELECT * FROM Invoice_Line_T;`

Transaction Management - Review Questions

1. The SQL statement
   `SELECT * FROM <table>;`
   is a transaction
   (a) True
   (b) False
2. A unit of work that changes the state of a database is a/an
   (a) trigger
   (b) stored procedure
   (c) embedded query
   (d) transaction
   (e) dynamic SQL query

3. Which statement undoes a transaction
   (a) COMMIT
   (b) UNDO
   (c) REDO
   (d) ROLLBACK
4. Which statement make changes to a database permanent
   (a) COMMIT
   (b) UNDO
   (c) REDO
   (d) ROLLBACK
   (e) SAVE
5. Transaction management is needed when:
   (a) a transaction consists of just one SQL command
   (b) there is a security risk
   (c) multiple SQL commands must be run as a single transaction
   (d) there is more than one database administrator
   (e) triggers are used

4.4 Range Check

It is possible to provide range check the the DDL level.

```
-- Construct to demostrate a date check in postgres at the DDL level.

DROP TABLE IF EXISTS Vaccine_T;

CREATE TABLE Vaccine_T(
FirstName VARCHAR( 25 ),
DOB DATE CHECK( EXTRACT ( YEAR FROM CURRENT_DATE ) - EXTRACT( YEAR FROM DOB ) > 10 )
);

INSERT INTO Vaccine_T( FirstName, DOB ) VALUES( 'Paas', '1997-12-21');
-- INSERT INTO Vaccine_T( FirstName, DOB ) VALUES( 'Fayel', '2012-02-25');

SELECT * FROM Vaccine_T;
```

A certain vaccine is available to patients above 10 years of age. `Paas` and `Fayel` try to get the vaccine. When the Date of Birth, `(DOB)` is entered, data for Paas goes through, but the system does not accept the data for Fayel. Uncomment the second `INSERT` statement, rerun the file and test the syntax. The function `CURRENT_DATE` fetches the date from the operating system.

Alternatively a CONSTRAINT can be added explicitly as shown below.

```
CREATE TABLE Vaccine_T(
FirstName VARCHAR( 25 ),
DOB DATE,
CONSTRAINT OVER_19 CHECK( EXTRACT ( YEAR FROM CURRENT_DATE ) - EXTRACT( YEAR FROM DOB ) > 19 )
);
```

Exercise: Put a constraint on `Prod_Price` to be greater than zero in `Inventory_DDL.sql` in the lab on page 114. Test your constraint by adding a product in `Inventory_DML.sql` file with a negative value or a zero. Then complete a short quiz on RANGE CONSTRAINT.

Thought Question: What would be an advantage in enforcing a constraint at the DDL level? It could have been enforced at the programming stage.

4.5 User Defined DataType - UDT

Maintaining and enforcing a consistent naming scheme for attributes is a challenge in a database system. Think of a large organization with many departments - manufacturing, sales, marketing, design among others. How would all departments use the same attribute name for `postal code` and they all enforced a six character length with the combination of letters and numerals in the Canadian postal system. A User Defined Datatype (`UDT`) is the answer. PostgreSQL `DOMAIN` is an object to implement a `UDT` with (or without) constraints.

The example in 4.5 creates a datatype called `Canada_PostCode_D` with a CHECK constraint of `A9A9A9`, character-numeral combination. This is implemented using a regular expression, the tilde ẽnforces case sensitivity.

```
DROP TABLE IF EXISTS Canada_Post_T;
DROP DOMAIN IF EXISTS Canada_PostCode_D;


-- domain will allow valid string of
-- characters and numerals as postcode used in Canada
CREATE DOMAIN Canada_PostCode_D AS TEXT
CHECK( VALUE ~'[A-Z][0-9][A-Z][0-9][A-Z][0-9]');

CREATE TABLE Canada_Post_T(
Address_ID SERIAL PRIMARY KEY,
Street1 TEXT NOT NULL,
Street2 TEXT,
City VARCHAR( 25 ) NOT NULL,
PostCode Canada_PostCode_D NOT NULL);
```

Figure 4.5: User Defined Datatype - UDT

Try the following insert statements to confirm the constraint.

```
DELETE FROM Canada_Post_T;


-- Successful INSERT operation
INSERT INTO Canada_Post_T( Street1, City, PostCode )
VALUES( '1385 Woodroffe Avenue', 'Ottawa', 'K2G1V8' );


-- Unsuccessful INSERT operations
INSERT INTO Canada_Post_T( Street1, City, PostCode )
VALUES( '1385 Woodroffe Avenue', 'Ottawa', 'K2g1V8' );


INSERT INTO Canada_Post_T( Street1, City, PostCode )
VALUES( '1385 Woodroffe Avenue', 'Ottawa', 'K21V8' );


INSERT INTO Canada_Post_T( Street1, City, PostCode )
VALUES( '1385 Woodroffe Avenue', 'Ottawa', '2K31V8' );
```

Thought Question: What would be the advantage of implementing such a constraint at the DDL level compared to implementing at the programming level or user level?

4.6 Function

It is easy to write a function in SQL. Although they are restrictive, i.e. SQL functions cannot have decision statements or loops, SQL functions can be used within SELECT statements. They cannot call UPDATE, DELETE or INSERT statements. Functions can call other functions. They are stored a database objects.

Figure 4.6 illustrates a simple function titled add_one, it accepts a number adds one and returns the result. Note the single quotes before BEGIN and after END;. You do not have name the passed variable as InValue, the parameter is then referred as $1 instead of InValue.

```
DROP FUNCTION IF EXISTS add_one( InValue FLOAT8 );

CREATE OR REPLACE FUNCTION add_one( InValue FLOAT8 ) RETURNS FLOAT8
AS '
BEGIN
RETURN( InValue + 1 );
END; '
LANGUAGE 'plpgsql';
```

Figure 4.6: SQL Function

The statements below illustrate its usage:

```
SELECT add_one( 7 );

SELECT add_one( 4.8 + 5.3 );

SELECT * FROM Employee_T, add_one( Bonus_Percent );
```

Bonus_Percent is an attribute in table Employee_T.

4.7 Stored Procedure

Similar to a function, a stored procedure is an object that is stored in a database and is used by all other database objects. Table

4.8 Trigger & Trigger Function

A trigger is an object associated with a table that runs a function when an UPDATE, INSERT or DELETE operation is performed on the table; i.e. trigger can be invoked when a data is changed. The function associated to a trigger must be written specifically for the trigger; it can call a general purpose function. A trigger must be associated to a table. A trigger function has access to data values *before* and *after* the data change event. These data values are used as NEW.<attribute> and OLD.<attribute>. OLD and NEW can be compared to determine if a change has taken place. A trigger function (although called a function) does not take any parameters nor does it return any parameters.

Production databases will have several triggers associated to tables. Some of the tasks they can perform are, data validation, audit, refresh MATERIALIZED VIEWs, send an email to a database administrator and other database maintenance tasks. A trigger can be invoked just once for all rows that have changed or for each row that has changed. Postgres allows only one trigger function to be associated to a trigger, if more than one function needs to run then multiple triggers must be created. Triggers associated to an event will run in alphabetical order. Data that has changed by an earlier trigger in the run order is available to subsequent triggers. A ROLLBACK on a later trigger will affect all previous triggers, they will also ROLLBACK. A trigger can be written in several languages, including plpgSQL, Python or Perl. *Caution:* Triggers can be used to enforce referential integrity constraints but this practice is not recommended, spurious use of triggers can degrade performance.

A trigger cannot be invoked from a `SELECT` statement; only from a data change operation, but a trigger can run a `SELECT` statement. Functions, stored procedures and trigger functions are stored in the database. Trigger functions are associated to triggers which in turn are associated to tables and are available only to tables even though they are stored in the database. Regular functions (i.e. non trigger functions) and stored procedures are also stored in the database but are not associated to a table, they are available to all objects in the database.

4.9 Comparison - Trigger, Function and Stored Procedure

The table below has been adapted from Database Processing, 40 Anniversary Edition, Kroenke et al., Page 378, Figure 7-33. It compares the three objects, Functions, Triggers and Stored Procedures.

	Function	**Trigger**	**Stored Procedure**
Accepts Parameters	Yes	No	Yes
Returns Result	Yes	No	Yes
Used in `SELECT` statement	Yes	No	No
Can use `SELECT` statememt	Yes	Yes	Yes
Uses `INSERT`, `UPDATE` and `DELETE` Statements	No	Yes	Yes
Can call a function	Yes	Yes	Yes
Can invoke a Trigger	No	Not Directly	Not Directly
Can invoke a stored procedure	No	Yes	Yes
Scope	as database object	specific to a table	as database object

Table 4.1: Comparison - Function, Trigger and Stored Procedure

4.10 Summary

This chapter has a few intermediate and advanced SQL topics that are covered in a later course in greater detail, they include triggers, stored procedures, embedded SQL, dynamic SQL, range constraints and user defined data types. In a sub-query, results of one or more queries are evaluated first, these results then serve as a parameter to another query. Correlated queries and non-correlated subqueries are differ in their execution. Care must be taken when writing correlated subqueries, they are CPU intensive when compared to non correlated subqueries.

5

Database Design

5.1 Objective

- Define a *relation*. Differentiate between a *table* and a *relation*. List properties if a *relation*.
- Define a *prime key*, *foreign key*, *composite key*, *surrogate key*.
- Maintain data integrity using *constraint*s.
- Define partial key dependency; identify and be able to remove it from a given relation.
- Define transitive dependency; identify and be able to remove it from a given relation.
- Define an anomaly with reference to a database.
- Identify the three anomalies in a database.
- Normalize data to third normal form (3NF) from unnormalized data.

5.2 Properties of a Relation

Six properties of a relation are:

1. Each relation in a database has a unique name
2. An entry at the intersection of each row and column is atomic. Stated differently, each cell has a single value
3. Each row is unique. Stated differently, no two rows are identical
4. Each attribute within a relation has a unique name.
5. The sequence of attributes has no consequence
6. The sequence of rows has no consequence

Two additional properties are

- Data in a column is the same kind. Columns contain data on attributes in an entity.
- Rows contain data on an entity.

5.3 Terms

Primary Key An attribute or a combination of attributes that uniquely identifies each row in a relation.
Stated alternately,
A primary key is the ***minimum*** number of attributes required to uniquely identify a row in a table.

Composite Key A primary key that consists of more than one attribute.

Foreign Key An attribute, or a set of attributes, in a table that has a corresponding data value in another table. *Referential Integrity* is ensured when data in the second table has a matching value in the first table. Usually, a

child table has a foreign key defined, this foreign key has a the same data value in the primary key of a parent table.

Surrogate Key A number assigned to uniquely identify a record. A surrogate key may serve as a prime key. It could be, a serial number with no significance to the data. A combination of letters and numbers such as a drivers licence number in Ontario, a health card number, a social insurance number (SIN) serves as a surrogate key. A key that is independent of user data is immune to changes in users data.

Constraint is a rule on data values, defined by a user, and enforced by the DBMS. A constraint maintains quality of data. Section 5.4 provides two examples of constraints.

Table Data represented in two dimensional format as rows and columns. As a convention, each column represents the same *type* (and same kind) of data.

A **Relation** is a two dimensional table of data that has the prime key defined, i.e. two rows of a relation cannot be identical. *Aside:* A relation is a special case of a table, all relations are tables, not all tables are relations.

Partial Functional Dependency A functional key in which one or more non-key attributes are functionally dependent on part, but not all, of the primary key.

Transitive Dependency In a relation, transitive dependency is present, when a non-key depends on another non-key.

Insertion Anomaly There are two types of Insertion Anomalies. First, data insertion in some attributes forces data into other attributes. Second, if data is not available for all attributes it forces a user to enter `NULL` values.

Deletion Anomaly other relevant data is possibly lost when a data element needs deletion.

Update Anomaly forces an update to all tuples when a change is made to one field in a tuple. Failure to update all tuples puts the database in an inconsistent state.

5.4 Constraint

Refer to `Inventory-DDL.sql` file, two examples of constraints are:

1. Primary key constraint, in the `Customer_T` table
   `CONSTRAINT PK_Customer PRIMARY KEY( Cust_Id )`
   This indicates that `Cust_ID` is a primary key. The DBMS will not allow a user to enter two customers with the same key. `PK_Customer` is a user defined object name. As a convention the prefix `PK_` is an aid to easily identify a prime key constraint.
2. In the `Invoice_T` table there are two constraints, the second constraint is
   `CONSTRAINT FK_Cust_ID FOREIGN KEY( Cust_Id ) REFERENCES Customer_T( Cust_ID )`
   The object name is `FK_Cust_ID`, it is a `FOREIGN KEY` constraint on `Cust_ID` in the `Invoice_T` table, each invoice will have a customer id (`Cust_ID`), the constraint will ensure that each entry in the `Invoice_T` table must (first) have an entry in the `Customer_T` table. It will not allow a user to add or change the customer id in the `Invoice_T` table unless it exists in the `Customer_T` table. Also, in the `Customer_T` table, an entry cannot be deleted if there is a record of the customer in the `Invoice_T` table.

5.5 Normalization

Normalization is the process of organizing the fields and tables of a relational database to minimize redundancy. Normalization usually involves dividing large tables into smaller (and less redundant) tables and defining relationships between them. The objective is to isolate data so that additions, deletions, and modifications to a field can be made in just one table and then propagated through the rest of the database using the defined relationships. [5]

The process of Normalization is reversible. For example, data from 3NF can be denormalized to 2NF, data from 2NF can be denormalized to 1NF and subsequently to UNF. No data is lost in the process. A normalized table is free from *insertion*, *deletion* and *update* anomalies.

A relation is in **First Normal Form** (1NF) when the following two conditions are satisfied

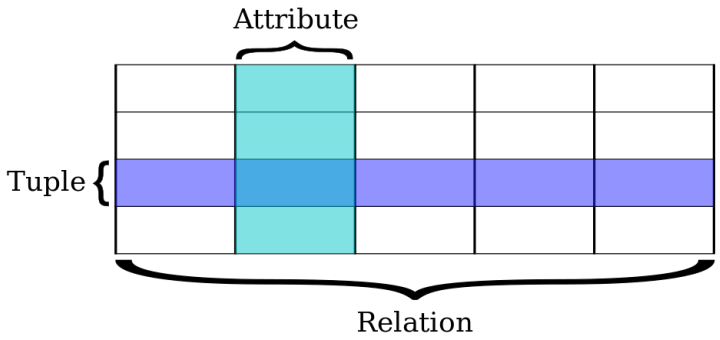


Figure 5.1: A Relation Ref: [6]

1. Repeating groups have been resolved, i.e. there is only one data value in an attribute. A set of multiple values are not permitted.
2. The primary key has been defined

Note: The term *primary key* is singular, even though it can be a combination of more than one attribute.

A relation is in **Second Normal Form** (2NF) when the following two conditions are satisfied

1. The relation is in First Normal Form
2. Every non-key attribute is fully functionally dependent on the primary key. Stated differently, all non-key attributes are fully dependent on the ***entire*** prime key.

A relation is in **Third Normal Form** (3NF) when the following two conditions are satisfied

1. The relation is in Second Normal Form
2. It has no transitive dependencies

Note: A table in 3NF is usually free from the tree anomalies.

5.6 Learning Activities

We shall identify anomalies and then normalize the tables.

5.6.1 First Normal Form

Exercise

Identify two disadvantages if data is represented as repeating groups.

Exercise

Study table 5.1. Identify the prime key. Give two reasons for your choice.

Exercise

Table 5.2 lists the planets and its satellites. Some planets do not have any satellites, some have only one and others have more than one. Put the data in First Normal Form by resolving repeating groups and identifying the

TeamID	Team
21	Woqag
23	Gavop
24	Turoz
25	Nibeg

Table 5.1: Team

prime-key. Do not insert additional attributes or surrogate keys, such as `ID`. Identify any shortcomings in the tables. State any assumptions you have made.

Planet	Satellite
Mercury	`NULL`
Venus	`NULL`
Earth	Moon
Mars	Deimos, Phobos
Jupiter	Ganymede, Callisto, Io, Europa
Saturn	Titan, Enceladus
Uranus	Titania, Oberon, Umbriel, Ariel, Miranda
Neptune	Triton

Table 5.2: Planet-Satellite Table

Exercise

Study table 5.3, its data is taken from the `world` database. Identify the prime key. In what Normal Form is the table? Give reason(s) for your answer.

5.6.2 Second Normal Form

We study tables with composite keys and normalize the data to Second Normal Form.

CountryCode	Country	CountryPopulation	CityID	City	CityPopulation
CAN	Canada	35540419	1814	Winnipeg	618477
CAN	Canada	35540419	1820	London	339917
CAN	Canada	35540419	1811	Calgary	767082
GBR	United Kingdom	63181775	456	London	7285000
GBR	United Kingdom	63181775	462	Manchester	430000
GBR	United Kingdom	63181775	464	Bristol	402000

Table 5.3: Country City Table

Exercise

1. Confirm that table 5.4 is in First Normal Form.
2. Is the primary key a composite key?
3. Give a reason why the table is not in Second Normal Form.
4. Normalize the data to Second Normal Form.

StudentID	CourseID	StudentName	CourseName	FinalGrade
704	8110	Suxot	Programming	A+
736	8110	Bimiq	Programming	A-
695	8001	Tigol	Math	B
695	8215	Tigol	Database	A-
798	8215	Jaceq	Database	B+
142	8001	Zarun	Math	A+
798	8001	Jaceq	Math	B+
142	8215	Zarun	Database	A
408	8110	Widaw	Programming	A

Table 5.4: Student-Course Table

Exercise

Study table 5.5 and answer the following questions.
Assume: A driver can drive more than one route.

1. Confirm that the relation is in First Normal Form.
2. Is the primary key a composite key?

3. Give a reason why the table is not in Second Normal Form.
4. What would happen to the data if DriveID: **653**, DriverName: **Binut**, is promoted to a manager and does not drive route **98**?
5. Normalize the data to Second Normal Form.
6. After Normalizing the data is the deletion anomaly resolved?

Route	**DriverID**	**DriverName**
95	485	Vizeq
95	658	Tutem
95	825	Semok
94	754	Fewup
94	658	Tutem
96	412	Sahab
96	825	Semok
97	157	Sonoq
97	570	Ziyir
98	653	Binut
99	773	Gitiv

Table 5.5: Route-Driver Table for OCTranspo

5.6.3 Third Normal Form

Study table 5.3. Will it be in First Normal Form after you have identified the prime key? Is it in Second Normal Form? Indicate why the table is not in Third Normal Form. Normalize the table to Third Normal Form.

5.6.4 Normalization Process

Figure 5.2 illustrates the process of bringing data from UNF to 3NF. The circle indicates a process, the rectangle represents data.

Figure 5.2: Normalization Process

5.6.5 Boyce-Codd Normal Form (BCNF)

BCNF is a more strict form of 3NF. In rare cases a relation in 3NF may show some anomalies, BCNF addresses these rare cases.

5.7 Guidelines on Normalization

Some general recommendations to follow in the normalization process.

General Recommendation

1. Read the Business Rules carefully.
2. Work on only one step at a time.
3. Read the Terms in section 5.3, page 59
4. Start with a clean sheet of paper, draw a logical ER diagram from rules provided.
5. Do not add any data.
6. Do not remove any data.
7. Do not add any attributes.
8. Do not eliminate any attributes.
9. The process should be reversible, i.e. it should be possible for data in 3NF to be brought back to UNF, using `JOIN`'s.

First Normal Form

1. Write the unnormalized data in a table.
2. Identify the prime key; the table is now a relation.
3. You may have to make simple assumptions. Write your assumptions clearly. Your assumptions should not contradict any business rules given to you.
4. Look for attributes which indicate ID's (Identification) such as CustomerID, `VIN`, `SIN`. These attributes are likely to be prime keys.
5. If the prime is only one attribute the relation is in 2NF. Stated differently, if the prime key is not composite, a relation in 1NF also in 2NF. A single attribute as a prime key cannot have any *partial-functional* dependencies.

Second Normal Form

1. Look for partial key dependencies.
2. How may attributes do you have from the first normal form? A single attribute as a prime key indicates that is already in 2NF.
3. If you have more than one attribute as a prime key, you need to identify those attribute that depend on part of the key.
4. Form tables from 1NF, with the prime keys already identified.
5. Your task is not complete, the newly formed tables will still have anomalies.

Third Normal Form

1. Work only on identifying non-keys that are dependent on other non keys
2. Break-up tables from 2NF to smaller tables, identify prime keys.
3. Verify that all dependencies are removed. Refine tables.

5.7.1 Normalization Exercise - Garage Shop

A vehicle repair-shop wants to keep records on vehicle repairs based on rules listed below. Normalize the database to 3NF.

Customer

1. A customer can have more than one vehicle
2. A customer brings in one vehicle at one time
3. Customer address is not recorded

Vehicle

4. Vehicle may be taken for a test drive; odometer reading may be higher when the vehicle is returned than when it is brought in
5. A vehicle can be serviced or maintained for more than one problem

Mechanic

6. A list of mechanics that work for the Garage-Shop is maintained
7. A vehicle can be worked on by only one mechanic during a repair incident
8. A mechanic can work on more than one vehicle in a day
9. A different mechanic can work on the same vehicle if it has more than one problem

Invoicing

10. Each instance of repair is recorded, with `DateTimeIn`, `DateTimeOut`
11. Cost of repair is not recorded (to keep simplicity in this exercise)
12. A vehicle taken in for repair is identified by `InvoiceID`
13. Each problem for an invoice has a `ProblemID`
14. `ProblemID` identifies the problem within an invoice. The count is reset for a new invoice
15. `InvoiceID` is unique for the company
16. Time when vehicle is brought in and returned is stored. Time is stored in 24 hour format

`CustomerID`, `CustomerName`, `CustomerPhone`, InvoiceID, `VIN, Make, Model, OdometerIn, OdometerOut, DateTimeIn, DateTimeOut, {ProblemID, MechanicID, MechanicName, Problem}`

`357, Varoq, 613-734-8712, IV-5291, RQ-3861, Toyota, Corola, 439876, 439880, 29-Oct-2014 11:29, 30-Oct-2014 15:30, {PR-01, S-211, Siv, Brake squeaking | PR-02, G-465, Gab, Noise in rear passenger wheel}`

`361, Tevif, 819-425-1329, IV-5292, LX-2478, Honda, Civic, 198176, 198176, 1-Nov-2014 9:27, 4-Nov-2014 12:30, {PR-01, X-191, Xad, Oil Change | PR-02, Y-532, Yob, Differential noise at low speed | PR-03, G-465, Gab, Change headlight bulb}`

Supplementary Exercises

1. In each table specify the number of rows the table will have, based on the sample data provided.
2. Write an SQL statement using `JOIN` clauses to simulate the results from 3NF to appear similar to your table in 1NF.

5.7.2 Normalization Exercise - Hotel Booking

A family owned hotel keeps records of his guests, employees and rooms. They take in guest bookings. Guests come repeatedly to their hotel. A small group of employees maintain and clean the room after the guests have left.

Draw a logical ER diagram and then a physical ER diagram from the information given. Show all cardinality. Chose appropriate names for entities and tables.

1. A guest can make more than one booking
2. A room can be occupied by a guest on many occasions
3. An employee maintains several rooms
4. A room can be maintained or cleaned by any employee

Hotel Rules:

1. Same `RoomCost` applies to all guests.
2. `BookingID` is unique for the hotel. Each room occupancy has a unique `BookingID`.
3. Sample data shows bookings and maintenance for two rooms.
4. Every time a guest leaves the room undergoes maintenance.
5. One maintenance is done per `BookingID`.
6. A new `BookingID` is given to a guest who books more than one room at the same time.
7. Only one employee cleans a particular room.

Aside: For simplicity, a convention if used for names, numbers and ID's

1. Guest names are chosen as five letters.
2. `GuestID` is coded as X-999, X is the first letter of the guests first name.
3. Employee names are four letters.
4. `RoomNumber` is three numerals; a room number is unique to the hotel.
5. `EmployeeID` is coded as X-99, X is the first letter of an employees first name.
6. `Checkin`, `Checkout` includes date and time. The format is yyyy-mm-dd, hh:mm, 24 hour time.

Normalize the database to 3NF. Write the column names for each normal form; do not show data elements.

```
RoomNumber, RoomType, RoomCost, { GuestID, GuestName, BookingID, CheckIn, CheckOut, EmployeeID,
EmployeeName, MaintenanceDate }
```

Sample Data:

```
235, Single BR, 112.00, { F517, Fiwix, 4, 2014-10-30 07:30:00, 2014-11-02 09:30:00, K35, Kiox, 2014-11-02
11:00 |  V385, Vitec, 7, 2014-06-21 01:30:00, 2014-06-25 14:00:00, P29, Peec, 2014-06-25 14:30:00
}


342, Suite, 180.00, { Q624, Qusar, 6, 2014-11-07 18:30:00, 2014-11-11 12:30:00, G59, Gaav, 2014-11-12
08:00:00 | M331, Meqit, 5, 2014-06-29 18:30:00, 2014-07-01 19:30:00, G59, Gaav, 2014-07-02 08:00:00
}


108, Double BR, 140.00, { F517, Fiwix, 12, 2015-01-07 12:30:00, 2015-01-09 15:00:00,
P29, Peec, 2015-01-10 08:00:00 }
```

5.8 Review Questions

1. The entity integrity rule states that:
   (a) no primary key attribute can be null
   (b) referential integrity must be maintained across all entities
   (c) a primary key must have only one attribute
   (d) each entity must have a primary key
   (e) each entity must have a foreign key
2. A foreign key must match a primary key value in another relation or the foreign key value must be `NULL`. This rule is called
   (a) Entity Integrity Rule
   (b) Referential Integrity Rule
   (c) Anomaly
   (d) Normalization
   (e) Functional Dependency
3. What value will you assign to an attribute when you do not know it's value or when the value is unknown
   (a) Use the same value from the previous record
   (b) Use the same value from the next record
   (c) Take the average values from the table and use it
   (d) Assign a `NULL` value
   (e) Remove the attribute from the table
4. An inconsistency may occur when attempting to update a table that is not normalized. What is the term given to such a situation.
   (a) functional dependency
   (b) transitive dependency
   (c) anomaly
   (d) domain constraint
   (e) recursive foreign key
5. A database is in First Normal Form when the following two conditions are satisfied
   (a) Repeating groups have been resolved and every non-key attribute is fully functionally dependent on the primary key
   (b) Repeating groups have been resolved and primary key has been defined
   (c) Repeating groups have been resolved and it has no transitive dependencies
   (d) Partial dependencies and transitive dependencies have been removed
6. A database is in Second Normal Form when Database is in First Normal Form and
   (a) repeating groups have been resolved
   (b) every non-key attribute is fully functionally dependent on the primary key
   (c) primary key has been defined
   (d) it has no transitive dependencies
7. A database is in Third Normal Form when Database is in Second Normal Form and
   (a) every non-key attribute is fully functionally dependent on the primary key
   (b) repeating groups have been resolved
   (c) primary key has been defined
   (d) it has no transitive dependencies
8. _____ is used for developing a prototype and testing queries.
   (a) Sample Data
   (b) Live Data
   (c) Backup Data

5.9 Supplementary Questions

Answer True or False

1. Normalization is done after Logical Database Design is complete.
2. Normalization takes into consideration how data is displayed, how it is used in reports and how a database is queried.
3. One of the two conditions for a relation to be in First Normal Form is: absence of repeating groups.
4. When inserting a row in a *normalized* database, i.e. 3NF, there may be additional insertions that result in duplication of data.
5. When deleting a row in a *normalized* database, i.e. 3NF, there may be loss of data.
6. When modifying a single row in a *normalized* database, there may be changes required to other rows.
7. The process of Normalization is reversible, i.e. it is possible to put data in UNF from 3NF.
8. In the Normalization process, it is possible that some data is lost.
9. A `NULL` value is the same as a space or a zero
10. The entity integrity rule states that a primary key attribute can be null.
11. Referential integrity is satisfied when every value of one column of a table exists as a value of another column in a different, or same, table.
12. The columns of a relation can be rearranged without changing the meaning or use of the relation.
13. Rows of a relation must not be interchanged and must be stored in a certain sequence.
14. A primary key is an attribute that uniquely identifies each row in a relation.
15. A composite key consists of only one attribute.

Select all statements that apply:

The purpose of Normalization is to

A. derive relations that are free of anomalies
B. simplify the enforcement of referential integrity constraints
C. make it easier to print reports, build queries and display data
D. make it easier to maintain data
E. pay attention to processing efficiency
F. pay attention on how data will be finally stored on physical storage

5.10 Prime Key Identification - Exercise

1. Identify the prime key in table 5.6 shown below.
   Is the key composite?

A	B	Cups
Carrot	Milk	1
Sugar	Sugar	3
Carrot	Honey	1
Pepper	Sugar	3

Table 5.6: Ingredients & Measurements

2. Identify the prime key in table 5.7 shown below.
   Is the key composite?

Animal	Diet	Weight
Horse	Hay	900
Horse	Oats	900
Grizzly	Deer	250
Grizzly	Berries	250
Elk	Vegetation	475
Moose	Vegetation	475

Table 5.7: Animal, Diet & Weight

3. Identify the prime key in table 5.8 shown below.
   Is the key composite?

Day	**Patient**	**Mood**
Mon	Waor	Happy
Mon	Yiur	Sad
Mon	Roup	Happy
Tue	Qaam	Grumpy
Tue	Wail	Tired
Tue	Waor	Happy
Wed	Waor	Joyful
Wed	Wail	Excited

Table 5.8: Day, Patient & Mood

4. Identify the prime key in table 5.9 shown below.

Pet	**Food**
Dog	Bone
Cat	Fish
Fish	Worm
Dog	Kibble
Horse	Hay
Dog	Bone
Rabbit	Cabbage
Horse	Carrot
Rabbit	Carrot

Table 5.9: Pet & Food

5. Identify the prime key in table 5.10 shown below.
   Is the key composite?

A	B	C	D
3	5	7	9
3	1	4	6
3	5	2	9
8	5	2	7
7	9	4	6

Table 5.10: Numbers

5.11 Further Normalization

Terms:

key: A combination of one or more attributes that uniquely identifies rows in a relation.

Candidate Key A key that determines all the other columns in a relation. Stated differently, a candidate key uniquely identifies a row in a relation. *Note:* A key although used in a singular noun, can have more than one attribute. *Aside:* It is possible to have more than one candidate key in a relation; i.e. it is possible to have more than one set of attributes that qualify as a candidate key. A key K_2 cannot be a candidate key if key K_1 is a proper subset of K_2.

Functional Dependency A value of one or more attributes determines the value of another attribute. Functional dependency is represented by A $\rightarrow$ B, which implies B is functionally dependent on A. Also, A determines B. Each value of A is associated with exactly one value of B. *Note:* A and B may consist of more than one attribute.

For a given value of A only one value of B can be found. When two tuples have the same value of A they must also have the same value of B. For a given value of **B** there may be more than one value of **A**. It is intuitive to define A and B in the direction of the arrow - A functionally determines B.

Determinant In the functional dependency is representation A $\rightarrow$ B, A is the determinant. Loosely, the attribute on the left hand side of the representation is a determinant.
A determinant is an attribute, or a set of attributes, that other attributes are fully functionally dependent.

Boyce-Codd Normal Form (BCNF) A strict form of 3NF. To bring a relation to BCNF data values, in a relation that is in 3NF, need to be examined. A relation in BCNF is also in 3NF; a relation in 3NF may not be in BCNF. A relation is in BCNF *iff* every determinant is a candidate key.
For a table to be in 3NF it is sufficient to remove partial functional dependencies from the prime key and then remove transitive dependencies considering the prime key. For a relation to be in BCNF partial functional dependencies must be removed from all candidate keys. 3NF can be achieved by an understanding of the attributes from business rules, to bring a relation to BCNF individual data values need to be analyzed. A relation can be further normalized to BCNF only if there are more than one candidate keys; i.e. a table that does not have more than one candidate key is guaranteed to be in BCNF.

Superkey A relation will always have at least one key. For example, a relation with three attributes `A`, `B`, `C`, will always have at least a key defined as `ABC`. This is called a trivial superkey. This property exists because in a relation, no two rows are the same.

Equivalent Terms: Table, Relation, File. Column, Attribute, Field. Row, Tuple, Record. Reference: Database Processing.

Exercise 1: Compare the definitions of **key**, **candidate key** and **prime key**. Note the differences. Can a table have

1. more than one candidate key?
2. more than one prime key?

Exercise 2: A table has `n` attributes. How many possible keys can it have?

Exercise 3: A relation has at least one candidate key. Is the statement true?

6

Physical Database Design and Performance

6.1 Objective

- Describe the three types of file organization
- Justify using an index structure
- Draw a simple diagram to explain an index structure
- Determine the appropriate index structure for an application
- Justify denormalization
- Improve database performance using horizontal and vertical partitioning

6.2 Index

An index is a *data structure* with a reference to data in the data file; it provides faster access to data. An index can be created on one or more columns. The data structure contains the attributes that are frequently used to searching; an index is considerably smaller in size compared to the actual data file. Indexes add an overhead in storage space and update time - they should be used with caution. Chapter 11 Indexes in `postgresql-9.5-us.pdf` provides further explanation.

> Indexes are usually created to satisfy a search criteria after the table has been in use for sometime and has grown in size. [8, Conn & Begg, Page 202]
>
> ...creation of indexes is *not* standard SQL. [8, Conn & Begg, Page 202]
>
> Indexes can be created only on base tables *not* on [dynamic] views. [8, Conn & Begg]

6.2.1 Creating and Deleting an Index

The general syntax for creating an Index is:
`CREATE [UNIQUE] INDEX Name ON <TABLE> <(Column)> [ASC|DESC]`
the default order is ascending.

Note:

1. The `Name` in the above statement is optional (in PostgreSQL). The DBMS automatically assigns a name to the index in for format <tablename>_<attribute>_idx
2. A comment may be added to an index (as in other objects). The syntax is
   `COMMENT ON INDEX cust_id_idx is 'Customer ID Index';`

3. A comment can be removed by `COMMENT ON TABLE cust_id_idx IS NULL;`

An index is removed using
`DROP INDEX [IF EXISTS] <IndexName>;`

Creating an index on more than one column will order the first column and then within the set of ordered rows the subsequent columns are ordered.

For example, an index on table `Passenger` on columns LastName and FirstName is created using
`CREATE INDEX LFName_IDX ON Passenger( LastName, FirstName )`

The results are shown in table 6.1

Last Name	**First Name**
...	...
Nibeg	Baj
Nibeg	Suk
Turoz	Viv
Turoz	Zec
...	...

Table 6.1: Passenger List

An index name must be unique within a schema, a DBMS does not allow reuse of index name within a database. In the example above, only one index with the name `LFName_IDX` can be created.

It is possible to create an index on part of the field, for example

```
CREATE INDEX PostCode_IDX ON Customer_T( SUBSTR( Cust_PostCode, 1, 3 ) );
```

6.2.2 Benefit and Overhead of Using an Index

An index stores the column values with address in a separate structure; a file structure. The addresses point to actual data values in the base table. When a query is initiated the search is made in the index file, the pointer (address) is matched with the entry in the base table and the row is extracted. Storing the column in an order, ascending or descending, is the sole reason an index speeds up a query.

Aside: Like an index at the back of a book - the keyword is associated with a page number, you first look at the index, which is in alphabetical order, then turn to the page number which contains the keyword. A search in the index is practical only because the indexed words are in alphabetical order.

An index is a file structure, actually a table, with a minimum of two columns the indexed column and the address of the records that contain the data; it occupies disk space. Each time an insertion, deletion or update is made to the base table the index needs to be updated. This is an overhead associated with maintaining an index. An index takes time to build, a table with *large* number of rows will take longer to build and index. Using an index slows updates, i.e. insertions, deletions and updates. A large number of indexes on different columns could degrade performance.

In practice, only small tables are without an index. Usually *large* tables will have an index atleast on the primary key. ***Large*** is a relative term; to determine if a table is large the supporting hardware on which the database resides needs to be taken into account; it includes hard disk size, CPU architecture, cache memory among other factors.

6.3 Sequential File

Data is stored in the order it arrives, but to view the data usually we need it in some order - alphabetical, numerical ascending, descending or chronological. In this section we shall explore different file structures used for data storage and retrieval. We shall also see how maintaining data in sorted order speeds retrieval.

6.3.1 Sequential Unsorted File

The first type, the simplest is a sequential unsorted file. Data is stored as it arrives. To search, the system goes through the list, compares each data element with the required key and stops until it finds the record.

Consider the list of cities in Peru, its District and population. Each time the user inserts a new city it is added to the bottom of the list. To search a city the database needs to go through the list from the first record to the last record, until the city is found.

City	**District**	**Population**
Piura	Piura	325000
Chincha Alta	Ica	110016
Arequipa	Arequipa	762000
Huancaya	Junin	327000
Castilla	Piura	90642
Juliaca	Puno	142576
Chimbote	Ancash	336000
Lima	Lima	6464693
Trujillo	La Libertad	652000
Ica	Ica	194820

Table 6.2: Cities in Peru, unsorted

To search for a record the best case is 1, i.e. the city you are searching for is at the top of the list. The worst case is n where n is the number of records. If there are 100 records, the database makes 100 comparisons. The average number of comparisons is n/2.

There is one advantage to this method, it does not take any extra space to maintain the table - just the space required to store the data elements themselves.

This method of storage and retrieval is used for small files.

6.3.2 Sorted List

We could extend this concept and keep the list in sorted order. To search a record in a sorted list the database will not go through the entire list, but will do a binary search. The same way you search for a word in a dictionary.

The best case to locate a record is 1 comparison, the worst case is log2(n) and the average case is also log2(n). In this sorted list there is still the advantage, no extra space is required to maintain the file. But there are other

disadvantages, when new data is added it goes to the end of the list, the list is not in alphabetical order. What happens when data is deleted or modified; again the list is not in alphabetical order. To benefit from an efficient search this file must be resorted each time data is added, deleted or modified. This method may possibly work when the data is static - additions, deletions and updates are infrequent and sorting the list occasionally is not a concern. For example, a list of provinces in Canada, this list does not change frequently.

City	**District**	**Population**
Arequipa	Arequipa	762000
Castilla	Piura	90642
Chimbote	Ancash	336000
Chincha Alta	Ica	110016
Huancaya	Junin	327000
Ica	Ica	194820
Juliaca	Puno	142576
Lima	Lima	6464693
Piura	Piura	325000
Trujillo	La Libertad	652000

Table 6.3: Cities in Peru, Sorted

City	District	Population
Arequipa	Arequipa	762000
Castilla	Piura	90642
Chimbote	Ancash	336000
Chincha Alta	Ica	110016
Huancaya	Junin	327000
Ica	Ica	194820
Juliaca	Puno	142576
Lima	Lima	6464693
Piura	Piura	325000
Trujillo	La Libertad	652000
Iquitos	Loreto	367000
Sullana	Piura	147361
Pucallpa	Ucayali	220866

Table 6.4: Cities in Peru, Three Cities Added - at end of list

The first ten rows are sorted, but the last three records that were added later are not in sorted. You need to sort the list all over again. All rows need to be moved. Each time data is added the tables needs to be sorted, this is not practical for large tables.

City	**District**	**Population**
Arequipa	Arequipa	762000
Castilla	Piura	90642
Chimbote	Ancash	336000
Huancaya	Junin	327000
Ica	Ica	194820
Lima	Lima	6464693
Piura	Piura	325000
Trujillo	La Libertad	652000

Table 6.5: Cities in Peru, Cities Deleted

The deleted rows in the table will have space taken up. To reclaim the space the table needs reorganization. All rows will need to be moved.

To summarize and compare the two methods as shown in table 6.6.

	Unsorted Sequential File	Sorted Sequential File
Advantage	Does not require additional disk space	
	Simplicity	Can perform a binary search, more efficient than sequential search
Disadvantage	Must perform a sequential search	Requires a sort each time data is updated, i.e. added, deleted or modified
Best Case	1	1
Average	n/2	$\log_2(n)$
Worst	n	$\log_2(n)$

Table 6.6: Sequential File Comparison

6.4 Hashing

The hashing method to search and retrieve a key uses an algorithm called *hash function*. The choice of the algorithm depends on the values of the keys. The hash function takes data (i.e. search key) as an input and generates a *hash value* or a ***hash***. This hash is then inserted into a table called a hash table. As new data arrives the table is built. Deletions and updates will automatically keep the table current. During lookup the process is repeated, the key is sent to the same hash function the hash generated and the table is referred. The hash is usually smaller than the key and is fixed length, even for keys of varying lengths. This method of search and retrieval is the fastest method among all other techniques. The computation time required to calculate the hash value is small. To create an index using hash use the following syntax in postgres.

```
CREATE INDEX [INDEX_NAME] ON <tablename> USING HASH <attributes>
```

For example, the syntax below creates an index on `Id` in the `city` table in `world` database.

```
CREATE INDEX CityID_IDX ON City USING HASH ( Id );
```

6.5 Comparison

A short comparison between the two index methods highlights their uses, advantages and disadvantages.

Description	**BTree**	**Hash**
Sequential Retrieval of Data	Rapid	Impractical
Random Retrieval of Data	Rapid	Faster than BTree
Deleting Records	Indexes need maintenance	Easier than BTree
Adding New Records	Indexes need maintenance	Hash on multiple attributes needs careful thought
Updating Records	Indexes need maintenance	Faster than BTree
Selecting a range of indexed data values	Naturally suited, since values are stored in order	Impractical, will take more resources than BTree
Maintenance	Usually automatic	Hash algorithm may be evaluation if data values are different or data volume increases

Table 6.7: Comparing Btree and Hash

6.6 Review Questions

1. In what phase of SDLC is indexing addressed?

   A. Logical Database Design
   B. Physical Database Design and definition
   C. Planning
   D. Analysis
   E. Implementation

2. The primary purpose of using indexes is to improve

   A. database performance during updates
   B. database performance during search and retrieval
   C. presentation of data in forms, reports and queries

3. Indexing on many columns

   A. has no effect on `UPDATE` operations
   B. has no effect on hard disk space usage
   C. has a tradeoff because each `INSERT` statement in the table needs to update the index as well

4. Smallest unit of data in a database is

   A. bit
   B. byte
   C. field
   D. record
   E. table

5. Identify the data type that accepts character data; only the required number of bytes are used in the storage medium

   A. `CHAR`
   B. `CLOB`
   C. `BLOB`
   D. `VARCHAR`
   E. `NUMBER`

6. Identify the data type that accepts fixed length character data of upto 2000 characters.

   A. `CHAR`
   B. `CLOB`
   C. `BLOB`
   D. `VARCHAR`
   E. `NUMBER`

7. A data structure used to determine the location of records in the actual table, or relation, that satisfies a condition

   A. normalization
   B. de-normalization
   C. index
   D. table space
   E. extent

8. Identify the file organization. The average number of comparisons to search a key is `n/2`, the maximum number of comaprisons is `n`, `n` is the number of records.

   A. index
   B. sequential file
   C. hash
   D. sequential, unsorted file

9. Identify the file organization that is least efficient when records are inserted, deleted and modified.

A. index
B. sequential file
C. hash

10. Which file organization is the most efficient to search on fields that are sorted

A. index
B. sequential file
C. hash

11. Identify the file organization where random retrieval of the key is the fastest.

A. index
B. hash
C. sequential file

12. Name given to an algorithm that is used to determine an address for data

A. index
B. sequence
C. hash

6.6.1 Additional Review Questions

1. Chose a data type that will store a medical X-Ray, MRI scans and other medical images

A. `CHAR`
B. `CLOB`
C. `BLOB`
D. `VARCHAR`
E. `NUMBER`

2. Database access frequencies are estimated from:

A. security violations
B. user logins
C. transaction volumes

3. A value that a field will store unless the user enters another explicit value; reduces data entry error, and saves time during data entry

A. range
B. `NULL` value
C. default value

4. A set of permissible values a field may accept

A. range
B. null value
C. default value

5. *Rows* of a logical relation is distributed to several separate tables, this procedure is called

A. normalization
B. de-normalization
C. horizontal partitioning
D. vertical partitioning

6. *Columns* of a logical relation is distributed to several separate tables, this procedure is called

A. normalization
B. de-normalization
C. horizontal partitioning
D. vertical partitioning

7. Identify the file organization that is most efficient when adding or deleting records

A. index
B. sequential file
C. hash

8. A table has 924 columns and the database is slow to process a query. The database administrator decides to split the table. In addition, a few columns contain sensitive data. She decides to isolate the columns to a separate table. What procedure should she use?

   A. first normal form
   B. second normal form
   C. de-normalization
   D. vertical partitioning
   E. horizontal partitioning

9. Currently, Algonquin College has all full time students and students in Continuing Education in one large table. It is not possible to accommodate the table on one hard drive. Database backup cannot be completed during after hours. The database administrator decides to split the table into two so that the students can be divided into two tables, full time students in one table and part time students in another table. What method does she use?

   A. normalize to first normal form
   B. normalize to second normal form
   C. horizontal partitioning
   D. vertical partitioning
   E. de-normalization

10. Ministry of Transportation Canada, has a large table for all vehicles in Canada. The database is normalized. It wants to store vehicle details based on the province or territory in which the vehicle is registered. What procedure should they use?

    A. normalize to first normal form
    B. normalize to second normal form
    C. horizontal partitioning
    D. vertical partitioning
    E. de-normalization

11. Health Canada wants only a few selected fields of patient data to be visible to a private company conducting medical research. What procedure should they use?

    A. normalize to first normal form
    B. normalize to second normal form
    C. vertical partitioning
    D. horizontal partitioning
    E. de-normalize the database

12. In horizontal and vertical partitioning the prime keys

    A. remain the same
    B. are different for each table
    C. depend on data values

Answer True or False

1. An index can be created on more than one column.
2. An index does not occupy any additional hard disk space.
3. It is possible that (many) indexes for a given table take up more hard disk space than the table itself.
4. A table `EMPLOYEE_T` has two columns:
   `EmployeeName, VARCHAR(25)`
   `Salary, DECIMAL(8,2)`
   The SQL statement

`SELECT AVG( Salary ) FROM EMPLOYEE_T;`
will be faster without an index.

5. Indexes are useful for *large* tables
6. An index can be created on more than one column, for example
`CREATE INDEX Customer_L_F_Name_IDX ON Customer( LastName, FirstName )`
is a valid statement.
7. Using an index reduces search and retrieval times, but increases time taken for updates.
8. In horizontal and vertical partitioning there is no change in access time nor any change in complexity
9. Queries can become more complex when database is normalized
10. Backup and retrieval is easier when the large files are partitioned horizontally and/or vertically.

6.7 Written Answers

1. What is clustering? When is it used?

2. What is horizontal partitioning? Give an example of horizontal partitioning.

3. What is vertical partitioning? Give an example of vertical partitioning.

4. The database design team in your organization is a group of three people, Dewip, Joxin and yourself? One of the designers, Dewip wants to denormalize a database. Joxin and you disagree with the idea. Dewip realizes denormalization will introduce anomalies, but is concerned with database perfomance and slow response time. What ideas will Joxin and you propose to convince Dewip to reconsider his thoughts on denormalizing. Divide your proposal into technical and business realities.

5. Indexing can be used on a materialized view but cannot be used on a dynamic view. Give a reason for the statement.

6. A table is used to store entries from a blog. Some of the fields are `username`, `date` and `comment`. The owner of the blog wants to limit the number of characters for comment to 1500 bytes. Suggest the data type to choose for the field `comment`. Justify your answer.

7. What is the smallest unit of data in a database?
8. During the October 2015 elections in Canada, there were 25,638,379 elibible voters. Elections Canada has stored all the voters in one sequential file that is sorted by a key, say Social Insurance Number (SIN). During the election held on 7th August 1867, the number of electors was 361,028. Voter list is maintained in two separate sequential files. To search a key, in each of the two data files, how many comparisons it would take for (i) best case, (ii) worst case and (iii) average case? What would be the values if the tables were not sorted on a key? Write your answers in the two tables below.

	1867 Elections			2015 Elections		
	Best	Average	Worst	Best	Average	Worst
Unsorted						
Sorted						

6.8 Lookup Table

A design technique that uses a simple table to store an *informal* code and a description. Consider the table `Part_T` table in lab on page 108. The `Material` attribute stores 9 characters for `Aluminium`, 5 characters each for `Brass` and `Steel`. A design decision could replace this column with one character, to store (say), `A`, `B` and `S`, for each of the materials use. An additional table then describes each of this characters. Table 6.8 illustrates such a table. For reporting purposes a simple `JOIN` will translate the material code into its description. It is easy to see the savings in storage space for (say) 100 or 1000 products. Using a lookup table is a design decision that is taken much later in the systems development life cycle. It is unlikely to show in the conceptual or logical design phase, it is also unlikely to show in business rules.

MaterialCode	Material
A	Aluminium
B	Brass
S	Steel

Table 6.8: Lookup Table. `MaterialCode_T`

Aside: What normal form is the lookup table in?

7

Modeling Data

7.1 Introduction

This lesson builds the foundation for Relational Databases. You are introduced to several new terms - they are used throughout the book. Modeling includes: Entities and Attributes, and their Relationships. Toward the end of the chapter Time Dependent Data Modeling is covered; this topic has relavance because of compliance regulations in US by Sarbanes-Oxley and Basel II and in Canada, Bill 198.

Considering the number of new terms and concepts introduced, we will revisit sections of this chapter several times. To focus our understanding on critical topics we will not introduce examples for certain concepts such as *degree of relationship*.

7.2 Objective

- Draw a Data Model from a give set of rules using entity relation diagram
- Business Rules
- List the types of attributes used in a DBMS
- Identify types of entities and relationships
- Define and differentiate *cardinality* and *degree*

7.3 Business Rules

Business Rules are the foundation of a data model, they represent the language and structure of an organization. They are derived from business policies, procedures, events, functions and other business objects. Business constraints are specified in business rules. Business rules provide a formal way to understand the organization by stakeholders. Some of the stakeholders are: owners of the organization, employees, information system designers in the organization. Business rules determine data in creation, storage, retrieval and data administration. Constraints are represented in Entity-Relationship Diagrams (ERD). Refer Page 54, [3].

7.4 Learning Activitiy

Use web resources to identify two *early* models of database that were used *before* relational model.

Is a `NULL` value different from blanks? What are the differences in queries when using `NULL` values

7.5 Relationships

In each of the ER diagrams fill in the entity of your choice that is appropriate to the the cardinality, label the relationship and enter a prime key and atleast one other attribute.

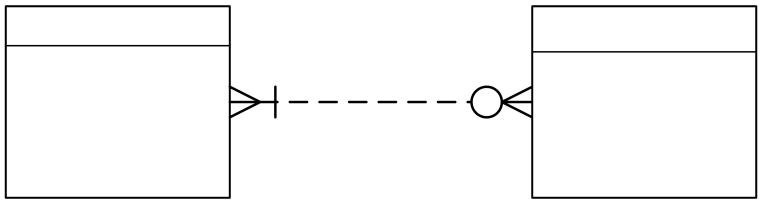


Figure 7.1: Many to Many

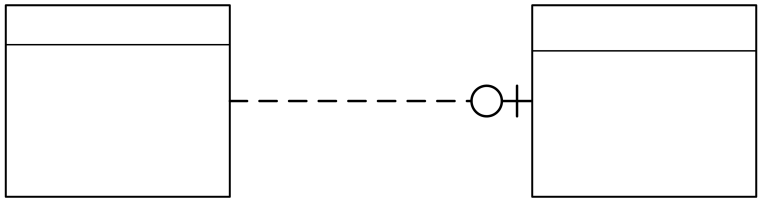


Figure 7.2: One to One

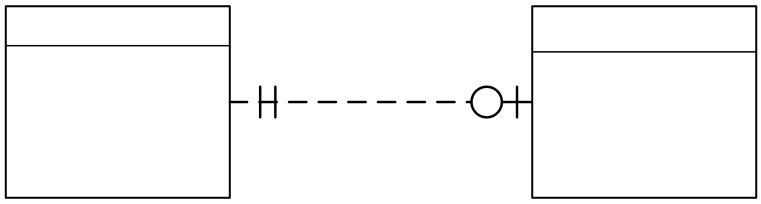


Figure 7.3: One to Mandatory One

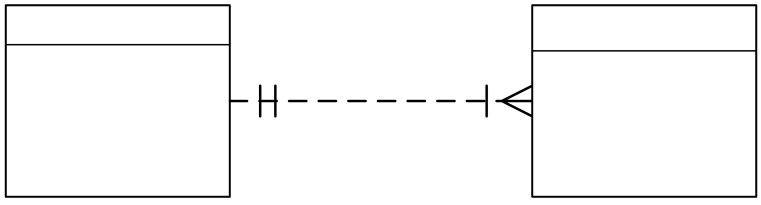


Figure 7.4: One to Many

7.6 Review Questions

Questions marked ™are from TestGen.

ER Model, An Overview

1. Logical representation of an organization's data is called a/an ™
   (a) Dataflow Diagram
   (b) Entity Relationship Diagram
   (c) Logical Diagram
   (d) The Business Rules Paradigm
   (e) Data Definitions

Modeling the Rules of the Organization

2. A Business Rule must be
   (a) Stored at multiple locations in a central repository but expressed only once
   (b) Stored only once in a central repository then shared throughout the organization
   (c) Stored at multiple locations in a central repository and shared throughout the organization
   (d) Stored only once in a central repository but never shared within the organization
3. A business rule can have many interpretations. Each stakeholder can have his own interpretation.
   (a) True
   (b) False
4. Data models in an organization change less frequently than business rules.
   (a) True
   (b) False
5. A word or phrase that has a specific meaning for the business
   (a) words
   (b) entity
   (c) term
   (d) relationship
   (e) business rule
6. A *fact* is an association between two or more:
   (a) words
   (b) entities
   (c) terms
   (d) relationships
   (e) business rules
7. `Supplier`, `Student`, `Book`, `Course` are examples of
   (a) Relationship
   (b) Business Rule
   (c) Entity
   (d) Degree of Relationship
   (e) Fact

Modeling Entities and Attributes

8. A strong entity is an entity that exists independent of other entity types
   (a) True
   (b) False
9. Identify the entity whose existence depends on another entity
   (a) Identifying Owner
   (b) Identifying Relationship
   (c) Entity Instance
   (d) Strong Entity
   (e) Weak Entity
10. An attribute that must have a value for every entity (relation) instance is a/an
    (a) Multivalued Attribute
    (b) Atomic Attribute
    (c) Composite Attribute
    (d) Required Attribute
    (e) Optional Attribute
11. An attribute that cannot be broken into smaller components is called a/an
    (a) Multivalued Attribute
    (b) Atomic Attribute
    (c) Composite Attribute
    (d) Required Attribute
    (e) Optional Attribute
12. An attribute whose value can be calculated from other attributes is called
    (a) Multivalued Attribute
    (b) Atomic Attribute
    (c) Composite Attribute
    (d) Derived Attribute
    (e) Optional Attribute
13. A Relational Database represents data as a collection of
    (a) Bits
    (b) Bytes
    (c) Attributes
    (d) Tables
    (e) Tuples
14. A time value that is associated with a data value, indicating when the data value was updated
    (a) Composite Value
    (b) Effective Date
    (c) Effective Time
    (d) Time Stamp
    (e) Compliance Regulation
15. The term `update` implies
    (a) Insertion of a data value
    (b) Deletion of a data value
    (c) Change in data value
    (d) Deletion or change in a data value
    (e) Insertion, deletion or change in a data value

16. Consider an invoice. `Line_Item_Amount` is an attribute that is the product of `Quantity` and `Unit_Price`, `Line_Total` is the sum of all `Line_Item_Amount` in an invoice. The data analyst decides not to store `Line_Item_Amount` and `Line_Total`. How will you describe this attribute?
    (a) Multivalued Attribute
    (b) Atomic Attribute
    (c) Composite Attribute
    (d) Derived Attribute
    (e) Optional Attribute

Questions of General Interest

17. E-R Diagrams were conceptualized by
    (a) C J Date
    (b) Edgar F Codd
    (c) Peter Chen
    (d) C J Date and Edgar F Codd
    (e) Donald D Chamberlin and Raymond F Boyce
18. Relational Database *theory* was developed by
    (a) C J Date
    (b) Edgar F Codd
    (c) Peter Chen
    (d) C J Date and Edgar F Codd
    (e) Donald D Chamberlin and Raymond F Boyce
19. Relational *Model* for database management was developed by
    (a) C J Date
    (b) Edgar F Codd
    (c) Peter Chen
    (d) Donald D Chamberlin and Raymond F Boyce
    (e) Charles Bachman
20. `Structured Query Language (SQL)` was developed by
    (a) C J Date
    (b) Edgar F Codd
    (c) Peter Chen
    (d) C J Date and Edgar F Codd
    (e) Donald D Chamberlin and Raymond F Boyce

Written Answer

1. Why is it important to store a business rule only once in a central repository?

2. Give an example of a weak entity and its corresponding entity. Write the necessary business rule(s) and assumption(s) to support your answer.

3. Define a table, or a set of tables, such that you can give an example of a *required attribute*, corresponding *optional attribute*, an *atomic attribute* and a *derived attribute*. Write the necessary business rule(s) or assumptions to support your answer.

4. How is a composite attribute different from a multivalued attribute.
   . Give an example of each.
   . What is the benefit of using a multivalued attribute? What could be possible disadvantages?
   . What notation is used to identify a multivalued attribute?

5. In relation to compliance regulations how is *time-dependent data* associated with sustainability. Which aspect of sustainability, can you relate to this type of data?

ER Modeling

1. Table 7.1 below has a multi-valued attribute `Hobby`, the prime key is `RegID`. It is difficult to search a person's hobby, and to count the number of people who pursue a hobby. Split the table into two tables to make search and count easier. Identify the prime key of each table.

RegID	**FirstName**	**LastName**	**Hobby**
8421	Tiwah	Xuyac	Ice Hockey, Badminton
1721	Kozev	Colec	Travel, Reading
9523	Vemox	Lewor	Orchids, Bonsai
1556	Lucim	Caros	Badminton, Reading, Travel
1038	Feres	Kohul	Ice Hockey, Yoga
1152	Nucom	Kowom	Yoga, Skiing

Table 7.1: Name-Hobby

7.7 Summary

Data modeling is an important step in the development process. A sound data model supports data integrity. Writing clear, consise and unambigious business rules is required for designing a database that is relevant to the organization. Although business rules change over time data models do not change at the same rate. This fact makes it imperative to put thought into developing a thorough data model.
There are several variations in ER diagram notations, we have adopted one variation - Crow's Feet.

Appendix A

Labs

A.1 Lab Submission Guidelines

Note: All items may not apply to every lab.

1. Save all `SQL` queries in a single, separate `.sql` file. `DDL` and `DML` statements should be in separate files.
2. Ensure commands in both files can be executed sequentially and independently. `DDL` statements should be executed first,the the `DML` statements.
   Aside: Keep the `.sql` file for revision (or study) during tests and exam.
3. Submit reverse, or forward, engineered ERD in `.png`, `.jpg` or `.pdf` format.
4. `CREATE VIEW` is a `DDL` statement, it should be in a `DDL` file. When you write a statement that will *use* a view, it is a `DML` statement. For example, `SELECT` statements that use a view should be in a query file, put these statements with the other queries you have written.
5. Business Rules and Abstract should be in `.docx`, `.doc` or `.rtf` format.
6. You have three attempts to upload to Brightspace, the last upload will be evaluated. If you upload a revised version, submit *all* files, not just the revised files.
7. Complete a quiz on the lab, if any.
8. Naming Convention: Name all files with eight character Algonquin email address, example `meaz0083-Inventory-DDL.sql`, for queries, `meaz0083-Inventory-ERD.pdf` for ERD, `meaz0083-Business Rules.sql` for Business Rules.
9. `JOIN` and `UNION` statemens should be with your queries, they are `DML` statements.
10. All explanations should be written in a separate `.txt` or `.rtf` file.

Appendix E

Navigating PostgreSQL

This page has been initiated by database students of Fall 2016. The following students have contributed to this page - Steven Adema,

E.1 Keyboard Shortcut

E.1.1 Edit

1. Comment a block of code `Ctrl-K`. Insert two dashes - as the first two characters on the line.
2. Uncomment a block of code `Ctrl-Shift-K`
3. Delete a line `Ctrl-L`
4. Duplicate a line `Ctrl-D`

E.1.2 Query

1. CTRL-E to open a query window. Or, click on the `SQL` icon
2. Run (Execute) a query `F5`

E.2 Metadata

1. Show all information of a table
```
SELECT *
FROM Information_Schema.Columns
WHERE Table_Name = 'country';
```
2. Show the column names of a table
```
SELECT Column_Name
FROM Information_Schema.Columns
WHERE Table_Name = 'country';
```
3. Show all objects in the database
```
SELECT *
FROM Information_Schema.Columns
WHERE Table_Schema = 'public';
```

E.3 Misc

1. to insert an apostrophe as a literal enclose the data item in double dollar signs. For example to add `St John's` as city name, use `$$St John's$$`.

E.4 psql

\? + ENTER KEY (backslash-question mark + ENTER)- to get a list of commands in `psql`

\c ⟨dbname⟩ dbname [username]; connect to database

\l list all databaeses on server

\dt list all tables in the current database

`SHOW DATA_DIRECTORY;`

E.5 Venn Diagram

Ian Glas has prepared these diagrams.

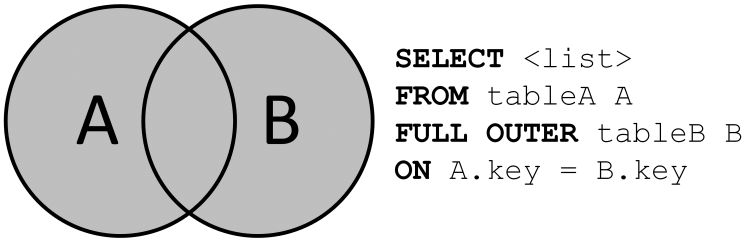


Figure E.1: Full Outer Join

Figure E.2: Full Outer Join using `NULL`

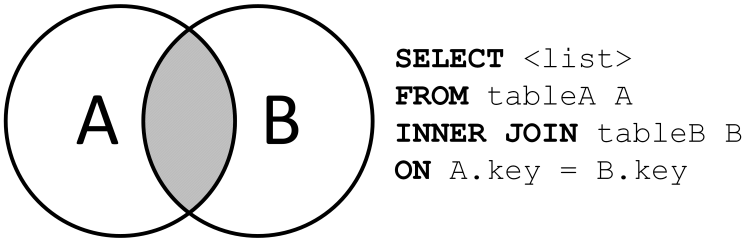
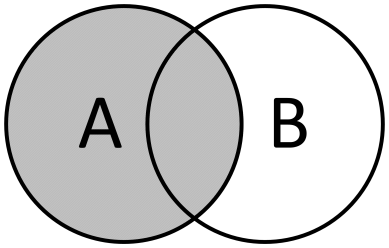
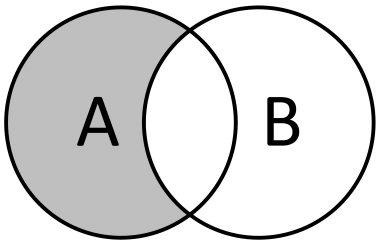


Figure E.3: Inner Join




```
SELECT <list>
FROM tableA A
LEFT JOIN tableB B
ON A.key = B.key
```

Figure E.4: Left Join




```
SELECT <list>
FROM tableA A
LEFT JOIN tableB B
ON A.key = B.key
WHERE B.key IS NULL
```

Figure E.5: Left Join using `NULL`

E.6 Link

A good resource for PostgreSQL tutorial `http://www.postgresqltutorial.com/` `https://www.techonthenet.com/postgresql/index.php`

E.7 Tips and Traps

`CREATE OR REPLACE TABLE` is not permitted. You have to first `DROP` the table and then `CREATE` it. Similarly `CREATE OR REPLACE TRIGGER` is not permitted. There is no method to accurately identify all dependencies of a `TABLE`.