MOUNT ROYAL
UNIVERSITY
1910

MOUNT ROYAL UNIVERSITY

Department of
MATHEMATICS AND COMPUTING

COMP 4522

Database II: Advanced Databases

Contents

EXERCISE

ONE

INTRODUCTION

Course themes
The importantce of data
Examples

In COMP 2521 you were introduced to the idea of a databases and learned about the relational database model and one of its implementations, MySQL. The focus of the course was to design and implement databases for transactional systems. Schemas consist of normalized tables and are designed to support ACID transactions.

In COMP 4522 we will be looking more at how data is used within an organization. That is, how can data be used for planning, how can data be used to make predictions, and then coming from that, how can we manage larger collections of data efficiently. The goal is to have students come away with an appreciation and understanding of the importance of data. We do live in the information age.

An example.

We are in the midst of a pandemic. Data is a vital part of government and health agency responses to COVID-19.

Each day AHS collects data from around the province on tests, positive tests, testing location, demographic data for those infected, hospitalizations, and deaths. (I think that in addition to deaths, the number of people who have severe protracted or chronic illness as a result of COVID should also be tracked. Similar to the car crashes, we report on the deaths, which are tragic, and if deaths are going down we think we are doing OK, but car crashes injure far more people than they kill, often with impacting people for the rest of their lives, which is also tragic.)

Each day the new data is provided to the province by the Chief Medical Officer of Health, Dr. Deena Hinshaw `https://twitter.com/CMOH_Alberta` and added to the provincial website `https://www.alberta.ca/coronavirus-info-for-albertans.aspx`.

Other people, like Robson Fletcher a CBC Calgary reporter, `https://twitter.com/CBCFletch`, take that information and provide a variety of analyses and visualizations. Analysis of the data is of vital importance. Is the curve exponential? Is it linear? Is it bending? Governments need to be able to enact protocols that are appropriate for the data and need to be able to see the impact of the restrictions being enacted. There is no point continuing to keep certain businesses closed if it is not changing the infection rate!

Notice that data is aggregated and (mostly) anonymized. From this web site it would be impossible to find out if your neighbour has COVID-19. There has been some controversy around the inclusion of age and ‘co-morbidity’ information. On the one hand, people should know who is at the highest risk, so they can take appropriate precautions, but on the other hand it can lead to people who think they are younger and healthier than they really are to take unnecessary risks.

Practise: Data Presentation and Visualization

Look at the data presented on the government web site and explore some of the graphs produced by `@CBCFlect`. Which are the most clear and useful? How easily can you find what you are looking for?

Now have a look at the Johns Hopkins Corona Virus Resource Center `https://coronavirus.jhu.edu/map.html`. This website provides an easy to read and clear summary of global information on COVID-19 in a variety of formats such as tables, maps, and graphs. While is it clear and seems authoritative the data was collected from a large number of sources, hundreds of different sources, that may use different definitions, different units, different frequency of collection, etc. Such information must be used cautiously.

Practise: Data Presentation and Visualization

Explore the Johns Hopkins Website. Think about the issues that the maintainers of this website face.

Another example, closer to home, is the MRU institutional analysis web-site: `http://ia.mtroyal.ca/`. This is good example of a *data warehouse* and visualization system. Data from the operational system is extracted, transformed and loaded (*ETL*) into the data warehouse and that data is accessible via this, rather primitive, dashboard.

In spite of the use interface this system provides data that is very useful for administering programs and courses.

Practise: Data Presentation and Visualization

Explore the Institutional Analysis website. Look of grade distributions for some of the courses that you have taken or are planning to take.

What are the key takeaways from these examples?

- You must know your data!
  - Where does it come from?

  - How was it collected?
  - How reliable is it?
  - When was it collected?
  - Who owns it?

- Privacy and security are important
  - Most data shown is aggregated and anonymized.
  - The grade data that is shown on the institutional analysis site does not include any student information. The course do not include instructor names.
  - The case data on the Alberta Health web site provides some demographic data (age, location etc.) but probably not enough to identify anyone.
  - Data should only be accessible to those that need and have a right to know it.
- What can you do/learn from/use this data for? And equally important, what *should* you use it for?
  - Awareness of privacy of data is increasing.
  - Bias in how data is presented is real.
  - The collection and use of data raises many ethical questions.
  - Data is complicated and the readers/audience may not understand the subtleties of curves and statistics. (While the term is thrown around a lot, many people don't really understand what exponential implies. How large a billion really is is also not well understood.)

Data drives many decisions that have a huge impact on our lives and our society and the response to COVID is only one important example. We must have good data and have confidence in the data that we are seeing.

This course is about *Collecting, Transforming, Storing, Analyzing, Predicting, Learning, Visualizing, and Interpreting.* data.

EXERCISE

TWO

RELATIONAL DATABASES

The database design lifecycle
Requirements
Conceptual Modeling
Logical Modeling
Physical Modeling
Transactions

In COMP 2521 you were introduced to the idea of a database and learned about the relational database model and one of its implementations, MySQL.

This chapter will give you a review of *Database I* in about 100 minutes. To do that I will use the database design lifecyle as a structure.

1. Requirements Gathering
2. Conceptual Modeling
3. Logical Modeling
4. Physical Design
5. Implementation
6. Maintenance

The database design life-cycle is not a linear process and there is no one methodology to follow. Some methodologies skip steps, start in the middle, and go back and forth. However thinking about the process this way is still useful as all of these things still need to be done, even if you are not doing them strictly in this order.

Remember that COMP 2521 was really focussed on transactional databases, that is the databases that support systems like Banner, the registration system used at MRU. Pretty much any non-trivial application uses a database to store information and 2521 gave you the basic tools needed to support such a system. COMP 3512 Web II and COMP 3504 Programming IV are good examples of using databases in an application.

Requirements Gathering

You learned about requirements gathering in systems analysis (COMP 2541). You gather documents and conduct interviews and create a set of business rules that describe your system. Business rules are an attempt to formalize requirements. The business rules drive the whole database design life-cycle process. Here are some examples of business rules.

- A student can register for a course if and only if the student has completed the prerequisite courses.
- A student is a person who has registered, now or in the past, in one or more courses.

- An active student is a student who is currently registered.
- An instructor is a person who teaches a section of a class, now or in the past. A person can be both a student and an instructor.

Notice that the rules are simple declarative sentences that talk about one thing at a time. They identify the terms — things we are concerned with and define the relationships between them.

Conceptual Modeling

Conceptual modeling is modeling the system and data as the user sees it The input to the process is a set of business rules, and the output is usually an Entity Relationship Model, consisting of an Enhanced Entity Relationship Diagram (EERD) and supporting documents.

EERDs are the accepted standard method for capturing a conceptual model. The supporting documents capture business rules that are not in the EERD, ambiguities, and assumptions made by the modeler.

ERDs where first introduced by Peter Pin-Shan Chen [**?**] in 1976. In the fast paced world of technology it is rare to have something stick around so long, but there are some good reasons for this:

1. Simple. An ERD consists of Entity Types and the Attributes that describe them and the Relationships between those Entity Types. That's it.
2. Flexible. Powerful. Can model many things easily. The diagrams are clear and easily understood.
3. Scalable. One can zoom in/out of the model allowing it to scale to large systems reasonably well.
4. Maps easily onto the relational model.

Remember that ERDs are not a Logical model! One can take the same ERD and convert it to different logical models: Relational, Key-Value, Hierarchical, or Document.

Logical Modeling

The logical model is modeling as the system presents it. The logical model may be related to the conceptual model, as ERDs are to the relational model, but they need not be. There are many logical models possible. Common ones are hierarchical, network, and relational. Edgar Codd first proposed the relational model in 1969 [?]. There are 12 rules that all relational database must follow, https://en.wikipedia.org/wiki/Codd%27s_12_rules, according to Codd. Since then it has become the standard for most database modeling. So much so that for many IT people database equals relational. Its strengths lie in its simplicity, its mathematical basis (relational algebra, which allows for theoretical study and practical optimization of queries), and its support for a query language (SQL). Other logical models are possible and in some cases, desirable. We will see later in the course that in a Big Data world where databases have billions of rows distributed around the world, the relational model breaks down and we need new models. Models that do not have a SQL type query language have become collectively referred to as NoSQL databases.

The Relational Model

Like the ER Model, the Relational Model is simple and consists of

1. Relations (In the mathematical sense, and not relationships)
2. Foreign Keys

It is important to understand what we mean by a relation, as we are using it here. It is a set of tuples:

$(d_1, d_2, \ldots, d_n)$

Where each d_j is an element of D_j a domain. Each tuple is an instance of something. For example, the elements of the tuple may be attributes describing a person (Name, Age, Weight).

Relations should have the following properties:

1. Each relation has a unique name;
2. A relation is a 2 d table consisting of rows (tuples) and columns (attributes);
3. Each cell has only one value in it;
4. Each column (attribute) has a unique name;
5. The order of columns does not matter;

6. The order of rows does not matter;
7. Every row is unique;
8. Values in a column all come from the same domain.

Relationships between relations are represented using foreign keys.

There are three Integrity Rules:

1. Entity Integrity. A relation always has a primary key consisting of one or more fields. Entity integrity is a different way of stating g) above. If every row is unique, there must be one or more fields in combination that are unique. The primary key can be a single field (like a student id) or some combination of fields (student id and CRN for a registration table).
2. Domain Integrity. Values in a column come from the same, well-defined domain.
3. Referential Integrity. (Special case of domain integrity). If a column is a foreign key, it must be the primary key in another table. In other words, the domain of a foreign key is the values of the primary key in another table.

Normalization

A sort of good reference is https://en.wikipedia.org/wiki/Database_normalization

Even if you follow all of the rules above, it is still possible to create bad relations. Normalization is a process to ensure that relations are well formed. The purpose of normalization is to minimize the possibility of anomalies.

- Insertion Anomaly — adding new rows forces user to create duplicate data
- Deletion Anomaly — deleting rows may cause a loss of needed data.
- Modification Anomaly — changing data in a row forces changes to other rows due to duplication of data.

Within a relation, there are relationships between the attributes. We refer to these relationships as functional dependencies, and there are three types:

1. Primary Key: All of the non-key attributes depend on the key.
2. Partial: Some non-key attributes depend on part of the key.
3. Transitive: A non-key attribute depends on another non-key attribute.

If a relation has any partial or transitive dependencies, it is possible that we may introduce anomalies when inserting, updating and deleting rows. The anomalies can be losing data or missing an update.

To avoid anomalies we break our relations up into smaller ones until only primary key dependencies remain. First, we remove partial key dependencies then we remove transitive dependencies. At each stage, our relations are in different normal forms.

- 1st Normal Form (1NF): No multi-valued attributes (no repeating groups). Primary key defined
- 2nd Normal Form (2NF) 1NF + All non-keys are identified by the whole key
- 3rd Normal Form (3NF) 2NF + All transitive dependencies are removed
- Boyce-Codd Normal Form (BCNF) 3NF + Every determinant is also a candidate key

In the end, we want the attributes to depend on the key, the whole key and nothing but the key. So help you, Codd.

SQL and Relational Algebra

Notice that there is no mention of SQL. SQL is part of the physical implementation, not the logical model. In the logical relational model, the relational algebra is used to perform queries. More about the relational algebra under separate cover.

Transactions

A database transaction is a group of database operations that form a logical unit of work — the transaction changes the database.

There are lots of things that can go wrong if we are changing the database. Something could crash in the middle of the transaction, one transaction could interfere with another etc.

Normally we want any transaction to conform to the ACID properties.

- Atomic All or nothing. All operations complete, or none do.
- Consistent: A transaction leaves the database in a consistent state. That is all integrity rules still hold.
- Isolated: When two or more transactions are operating concurrently they must not interfere with each other (intermediate results are not visible).
- Durable: The result of a completed transaction is permanent. For example, if a transaction completes and the system immediately crashes, the results of the transaction should still be there when the system restarts. You need to be clever to make this one work.

Physical

The physical implementation is done using whichever platform you have at hand. There are a number of ISO standards for SQL implementations and most implementations are similar enough that a developer will not have any trouble moving from one to the other. Key players in the area are Oracle, `https://oracle.com`, MySQL,`mysql,com`, which Oracle picked up a few years back, SQL Server, `https://www.microsoft.com/en-us/sql-server/sql-server-downloads`, Access and others.

RDBMSs provide a variety of services that normally will include:

- DDL — Data Definition language — A way to create, delete and modify tables, views, triggers and procedures and other database objects.
- DML — Data Manipulation language — A way to insert, delete, update and retrieve data from tables. This is SQL.
- A catalogue that can store information about what is in the database — a data dictionary. One of Codd's rules is that the data dictionary must itself be relational tables.
- Support for transactions.

- Support for backup and recovery.
- A mechanism to control access.

Key Points

- The relational model with normalization and transaction support both ensure that the database remains in a consistent state. For a transactional database, this is what we want! For many applications, such as student records system or a banking system, it is vital!

  We will see later that for data warehousing and in other “Big Data” applications (think Facebook, Amazon, etc.) we may want or need to relax that strict consistency for the sake of availability.

EXERCISE

THREE

RELATIONAL ALGEBRA

Definition
Relational Operators

THE RELATIONAL DATABASE model is based on the *relational algebra*. One of the reasons that relational databases are so sucessful and widely used is that this mathematical under-pinning facilitates analysis and optimization.

Remember: A *Relation* is a 2d table of data with certain properties:

1. The relation has a unique name.
2. Each column (attribute) has a unique name.
3. Each row in the relation is unique. This is equivalent to saying that there exists a field or fields that are a primary key. It also means that there are no duplicate rows.
4. The order of the rows does not matter.
5. The order of the columns does not matter.
6. The values in each column come from the same domain.
7. There can be only one value in each cell (intersection of row and column).

You are all familiar with numbers. For simplicity, think of the set of integers. If you have some integers, you can use arithmetic operators like

+ addition

- substraction

$\times$ multiplication

$\div$ division

– negation

to create new integers!

For example, you have the integer 6. You can create a new number by negating it:

$-(6)$

Notice that negation only takes one operand, it is a *unary* operator. Or if you have 4 and 6 you can add them to create a new integer

$+(6, 4) = 10$

Notice that addition takes two operands, it is a *binary* operator. We can also define new operators based on these basic ones. For example the i operator which is a unary operator that adds one to an integer.

$i(a) = +(a, 1)$

You are probably more familiar with inline notation, but this notation is equivalent.

Similarly, the relational algebra is a way of manipulating relations to create new relations.

There are five primitive operations (and one more that helps) that operate on *Relations* to produce new relations

1. Projection Π
2. Selection σ
3. Cartesian Product $\times$
4. Union $\cup$
5. Difference $-$
6. Rename ρ

Operations are either *unary* or *binary*. Unary operators take a single relation and yield a single relation. Binary operators take two relations and yield a single relation.

For the examples, consider three relations A and B and D. A has attributes a, b, c, d, B has attributes m, n, o and D has attributes p, q, r. They can be written as $A(a, b, c, d)$ and $B(m, n, o)$ and $D(p, q, r)$.

Relational algebra operations always produce valid relations. So if an operation would produce duplicate rows, they are removed and only the distinct set of rows remain. Similarly for attributes.

1. Projection Π

   Unary.

   The result is a new relation with the same number of rows, but only the subset of attributes specified.

   Example:

   $C = \Pi_{c,d}(A)$

   The projection yields a new relation C that has all of the rows of A, but only the attributes c, d, i.e. $C(c, d)$.

   Projection is like taking a vertical slice through the relation.

2. Selection σ_ϕ

   Unary.

The result is a new relation with all of the attributes, but those tuples (rows) that satisfy the given proposition, ϕ.

ϕ can be constructed with simple propositions like "NAME = Jane" or can be a compound construction using AND, OR, NOT.

Example:

$C = \sigma_{a<100 \text{AND} a>10}(A)$

The selection yields a new relation C with attributes a, b, c, d, e and only those rows where a has a value greater than 10 and less than 100.

A selection is taking horizontal slices of the relation.

3. Cartesian product $\times$

   Binary.

   This is a slight variation on the familiar set Cartesian product. The result is a new relation with tuples made up of all the attributes from each relation.

   $C = A \times B$

   The product yields a new relation C with tuples composed of all attributes from A and all attributes from B. Each tuple in A is matched with each tuple in B, this means that the cardinality of the result, $|A \times B| = |A| \times |B|$

   C can be expressed as $C(a, b, c, d, e, m, n, o)$.

4. Union $\cup$

   Binary.

   Result is a relation with all tuples from A and B. Duplicates are removed (because a relation cannot have duplicates).

   A and B must be union compatible (the same number of attributes with compatible domains),

   Example:

   $C = B \cup D$

   The union would create a new relation $C(m, n, o)$ will all the rows from B and D.

5. Difference $-$

   Binary.

   $B - D$ Result is a relation with all tuples in B but not in D. B and D must be union compatible.

Example:

If B has rows (tuples):

2	Tree	4.5
3	Cat	9.6
6	Sparrow	1.6

and D has rows

2	Tree	4.5
8	Tomato	5.4

then $C = B - D$ has rows

3	Cat	9.6
6	Sparrow	1.6

and $C = D - B$ has rows

8	Tomato	5.4

6. Rename ρ

   Unary. Result is identical to A except with attributes renamed.

   $\rho_{(a/b)}(R)$

These operations can be used to construct more complicated operations. Operations can be nested.

With these five operators you can construct any relational query. For example, you can combine $\times$ and σ to create an inner join $\bowtie$.

Why?

Allows analysis and optimization. You can build a tree with leaves that are relations, nodes are operators and subtrees are sub-expressions.

Simple optimization: Minimize size of relations and find common sub-expressions.

The relational algebra is procedural. You follow the steps.

Practise

1. Given a logical database schema for storing information on books:

```
B = BOOK (isbn, pubDate, listPrice, publisher, title, copyright)
   PK = (isbn)

A = AUTHOR (firstName, lastName, address)
   PK = (firsName, lastName)

BA = BOOKAUTHOR (isbn, firstName, lastName)
   PK = (isbn, firsName, lastName)
   FK (isbn) ==> BOOK(isbn)
   FK (firstName, lastName) ==> AUTHOR(lastName, firstName)

S = BOOKSTORE (name, address)
   PK = (name)

I = INVENTORY (storeName, isbn, storePrice, quantity)
   PK = (storeName, isbn)
   FK (storeName) ==> BOOKSTORE(name)
```

Assumptions:

- Author lastName and firstName uniquely identify an author.
- Books can have multiple authors.

Write relational algebra expressions for the following queries:

(a) Return all books (isbn only) with a list price over $50.

(b) Return all books (isbn only) with a publish date before Jan. 11, 2001 or whose publisher is ’GenCo’.

(c) Return all addresses (both for authors and bookstores).

(d) Return all authors that have not published a book.

(e) Return the list of books written by Joe Smith.

(f) Return the list of books written by Joe Smith that cost less than $40.

(g) Find all authors (firstName, lastName) who have written books that have been published after Feb. 15, 1990.

(h) Find all authors who have written more than one book.

(i) Find pairs of books with different ISBNs but the same title. A pair should be listed only once; e.g., list (i,j) but not (j,i).

(j) List all the books (isbn only) that bookstore 'All Books' sells that bookstore 'Some Books' also sells.

(k) List all the books (isbn only) that bookstore 'All Books' sells that bookstore 'Some Books' sells for less.

2. Given a set of relations

```
Gilligan's Island Castaways

Castaway(C_ID, CName, Email, DaysStranded)
   Primary Key: C_ID

Hut(H_ID, HName, Location, FloorArea)
   Primary Key: H_ID

CastawayHut(C_ID, H_ID)
   Primary Key: C_ID+H_ID
   Foreign Key: C_ID -> Castaway
   Foreign Key: H_ID -> Hut
```

(a) What is the result of the following relational algebra expression?

$$\sigma_{DaysStranded>100}(\Pi_{CName,HName}(CH \times H \times C)) \quad (3.1)$$

Where CH refers to the CastawayHut relation, H to the Hut relation and C to the Castaway relation.

(b) What is the result of the following relational algebra expression?

$$\Pi_{CName,HName}(\sigma_{DaysStranded<100}(C)) \quad (3.2)$$

Where CH refers to the CastawayHut relation, H to the Hut relation and C to the Castaway relation.

(c) Write a relational algebra expression which will display the castaway name, and the name of the hut that they live in.

3. Write a general relational algebra expression for an inner join, $\bowtie$.

4. Write a general relational algebra expression for intersection, $\cap$, the rows that are common to two relations.

5. Write a general relational algebra expression for a left outer join, $⟕$.

EXERCISE

FOUR

RELATIONAL ALGEBRA II

Extended Relational Algebra
Aggregation

YOU MAY HAVE NOTICED something lacking in the description of relational algebra in the previous section 3. How can we do a *Count* or *Max* or *Average*?

First here is a formal definition of expressions in relational algebra:

A basic expression in the relational algebra consists of one of the following:

- A relation;
- A constant relation;

A constant relation is specified by writing its tuples within set braces

$\{(22222, Einstein, Physics, 95000), (76543, Singh, Finance, 80000)\}$

A general expression is either a basic expression, or can be constructed from other general expressions.

If E_1 and E_2, are relational algebra expressions, then the following are also relational algebra expressions.

- $E_1 \cup E_2$
- $E_1 - E_2$
- $E_1 \times E_2$
- $\sigma_P(E_1)$, where P is predicate on the attributes of E_1
- $\Pi_S(E_1)$, where S is list of attributes of E_1
- $\rho_x(E_1)$, where x is the new name for E_1

Aggregate Operations

We can also define the aggregate operation, $\mathcal{G}$.

Aggregate functions take a collection of values and return a single value.

For example, if we have a relation $R(val)$

val
1
1
3
4
4
11

then

$\mathcal{G}_{sum(val)}(R)$

would return a relation with a single row with the value 24.

We can define a variety of aggregate functions, such as *max*, *min*, *count*, *average* etc.

Consider the relation EMPLOYEE

ID	Name	DeptName	Salary
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

We can find the average salary of all instructors with a query like:

$\mathcal{G}_{avg(Salary)}(EMPLOYEE)$

What if we wanted to show the average for each department? We can list the attributes that we would like to group on before the $\mathcal{G}$.

$DeptName\mathcal{G}_{avg(Salary)}(EMPLOYEE)$

Which will return a relation like this:

DeptName	AvGSalary
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

The general form of the aggregate operation is

$$G_1, G_2, ...G_n \mathcal{G}_{F1(A_1), F2(A_2), ..F_m(A_m)}(E)$$

E is any relational algebra expressionA, $G_1, G_2, ...G_n$ is list of attributes to group on; Each F_i is an aggregate function and each A_i is an attribute.