1 Database management systems

Databases are developed in a way to grant data independence, integrity and security, efficient and concurrent access and reduced application development time.

1.1 Data models

A data model are the concepts used to describe data. A schema is the description of a particular collection of data.
The most used model today is the relational model. It bases its self on the concept of relations, and every relation has a schema

1.1.1 Abstraction

Data is defined in different schemas. The physical schema represents how the data is stored on the disk, the conceptual schema defines the logical structure, and a view is how users see the data.

Schemas are defined using DDL (Data Definition Language) and the data is modified using DML (Data Manipulation Language)

1.2 ACID

ACID is the list of properties that a database must ensure. To explaine them we use the example of a bank transaction

1.2.1 Atomicity

Atomicity is a property caracterized by the all-or-nothing policy. It is applied through a log who keeps

1.2.2 Consistency

The database must be consistent before and after the transaction

1.2.3 Isolation

multiple transactions occur independently without interference

1.2.4 Durability

THe changes of a successful transaction occurs even if the system failure occurs

2 E/R Model

2.1 Entity

an entity is a real world object which is described using a set of attributes. A collection of similar entities is a entity set

weak entities

This are entities that depend to another entity. The weak entity doesn't make sense to exist by itself. This entities are rappresented through a thick line around the entity and they use a thick arrow towards the entity they depend

2.2 Relations

It is a connetion between two or more entity sets.
The relation can use different multiplicity, which are many to many, many to one and one to one.
We use the wisconsin representation system, which is structured like this:
we use an arrow to point to "one", a think arrow to represent "exactly one", a line to represent "mmany to many" and a thick line for "at least one"

2.2.1 Aggregation

is used when we have to model a relationship involving entity sets and a relationship set

2.3 Subclasses

it's a special case of subentity, it's represented with a triangle

E/R Subclasses

Subclasses form a tree of one-one inheritance that use isa relationships.

2.4 Keys

This are unique identifiers of an entity and are represented through an underlined attribute. They can be super keys (keys in general), candidate key (minimal super key) and primary key (the chosen key)

2.5 Schemas

3 Relational model

it's the most used model thanks to the characteristic to be easier and faster than other models.

Definitions

a relational database is a set of relations, which are made of instances, or tables, and schemas, which specifies name of the relation, and name plus type of each column. In the tables, rows equals to the cardinality and are called tuples,the fields are called degrees

3.1 Integrity contraints

A condition must be true for any instance of the database.

Keys can be contraints because no distinct tuples can have the same values

4 Relational algebra

The manipulation and retrieval of data from a database is managed through query language. A mathematical query language i the relation algebra, an operational query language.

The basic operations are:

- Selection (σ): selects a subset of rows from relation
- Projection (π): selects the columns from relation
- Cross-product ($\times$): allows to combine two relations
- Set difference (-)
- Union ($\cup$)

there are also other sub-operators:

- Renomination (ρ, $\rightarrow$)
- Join ($\bowtie$) or theta join: the union on determined conditions
- Equijoin: a join on only equalities
- Natural join: is a equijoin on all common fields

5 SQL

A basic SQL query is composed by relation-list, target-list and qualification and it follows this strategy:

- Compute the cross-product of relation-list (FROM)
- discard resulting tuples if they fail qualifications (SELECT)
- delete attributes that are not in target-list (WHERE)

SQL is also capable of nesting queries. This is done by adding the operator IN in the WHERE section, and the nested query inside. It can be also used the operator EXISTS to let the inner query use parameters from the outer query. parentesis

5.0.1 Operators LIKE and AS

LIKE is used in string matching, it uses % to represent multiple characters and _ just for one. To generate new name fields is also used the operator AS

5.0.2 UNION and INTERSECT

UNION can be used to unite two compatible sets of tuples, meanwhile INTERSECT computes the intersection between the tuples

5.0.3 Agregate Operators

SQL offers various extentions to relational algebra, such as COUNT, SUM, AVG, MAX and MIN

5.1 Deductive Databases

TO apply recursevly

6 Indexing

On datastorage, Disk can retrieve random pages at fixed cost, but must be organazied. This is called file organization, or the mothod of arranging a file of records. Indexes are the data structures that allow to find records ids.Indexes are organized through trees or hashing.
Alternatively, other file organizations can be heap files or sorted files.

An index contains a collection of data entries $x*$ with a key value of x. Any field of a relation can be the search key for an index.

6.0.1 B+ tree indexes

Data entries are contained in the leaves of a tree and are chained to one another. Non-leaf pages have index entries used to direct searches

6.0.2 Hash-based indexes

Better suited for equality selection, this indexes are a collection of buckets (primary pages with zero or more overflow pages) which contain data entries. It's also implemented an hashing function that retrieves the buckets where a record belongs.

6.1 Alternatives for data entry x in index

we can store the data record with key value x, rids of data record or list of rids of data records.

In the first alternative, the index structure is a file organization of data records. At most one index on a collection of data records can use this alternative, otherwise the data is duplicated. If data records are very large, the number of pages is high, this implies also lage size of the auxiliary information

With the other alternatives data entries are much smaller than data records, so more efficient for large data records. The third alternative is even more compact

6.2 Clustered and unclustered indexes

An index can be categorized in primary vs secondary, and clustered vs unclustered. These second classification differs for a sorted or unsorted heap file with all the data records

7 Cost Teory

Definitions:

- Page: Is the size of one unit inside a hard drive. It's denoted with p
- Record size: is the size of a tuple, denoted with t_R
- Pages of relations: is the number of pages occupied by the records of a single relation. it's denoted with P_R
- Cardinality of a relation: is the number of records of a single cardinality, is denoted with $|R|$
- Cardinality of an attribute: is the number of distinct values inside an attribute. Is denoted with $R.A$

This definitions can be used to determin other data

$$\text{\# Records of } R \text{ per page:}$$
$$\lfloor \frac{P}{t_R} \rfloor = \lfloor \frac{PageSize}{t_{TupleSize}} \rfloor$$

$$\text{Size of R:}$$
$$\lceil \frac{|R|}{\lfloor \frac{P}{t_R} \rfloor} \rceil \times p = \lceil \frac{\#Tuples}{\#TuplesPerPage} \rceil \times p$$

7.1 Cost

Every action of I/O on the database requires time, by definitiono we will say that an action of writing and reading will cost 1. The other tipes of action are:

Scan

Is the operation of reading a relation, his cost is equal to the pages of a relation

Sorting

The cost of sorting the data depends on the number of memory buffers that the database can use. If only 3 buffers are avaiable the cost is equal to:

$$2 \times N \times (\lceil log_2 N \rceil + 1)$$

where N is the number of pages that a relation occupies.
In case more than 3 buffers are avaiable then the cost is

$$2 \times N \times (\lceil log_{B-1} \lceil \frac{N}{B} \rceil \rceil + 1)$$

where B is the number of buffers avaiable

Index lookup

the cost of using an index is denoted with L, and is equal to 1.2 in a hash index, meanwhile, for a B-tree the cost equals to $log_s|R.A|$, where s equals to the maximum amount of children a node can have.

The retrieval of unclustered data equals to $L + |R_{A=x}|$, meanwhile the retrieval of clustered data is $L + \frac{|R_{A=x}|}{\lfloor \frac{P}{t_R} \rfloor}$ or $L + \#pages\ occupied\ by\ the\ tuples$

Selectivity factor

This is the amount of records expected to respect a given condition. These records are denoted as Rc. The selectivity factor is denoted as:

$$f = \frac{|R_{A=x}|}{|R|}$$

Update

The cost equals to the index lookup plus the reading and writing of the page, or

$$L + \#pages\ with\ the\ updated\ records \times 2$$

7.2 Joins

We will use the relation R and S

Nested loop join

The cost equals to $P_R + P_S \times P_R$

7.2.1 Sort Merge Join

The cost equals to the Cost of sorting R plus the cost of sorting S plus $P_R + P_S$

Hash join

This join uses a hash map to make the action more efficient. The cost equals to $3 \times (P_R + P_S)$

Index nested loops join

This method uses the index of the second relation to make the join more efficient, if the index doesn't exist, then this method is not usable. The cost equals to:

$$P_R + |R| \times (\text{index lookup cost + cost of retrieveng the qualifying records})$$

7.3 Query plans

A query can be converted in tree composed of relational algebra operations, which is used to compute in advance the cost of the query. Every query can generate different trees, so it's the database role to determine the most efficient. This action is called optimization.

Some operations don't need to wait for the final result, but they can use records as they arrive. This operations are called **On-the-fly** operations. Other operations don't even need to use records but only their indexes, so they are called **Index-only** operations

8 Functional dependencies and normal forms

8.1 Functional Dependencies

In a relation, it happens frequently that some information is repeated in a pattern. In this case we say a specific attribute functionally determines the values of other attributes. This constraints are called functional dependecies.

More specifically, a functional dependency is expressed as $X \to Y$ if $X, Y \subseteq U$ and if $t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$

8.2 Decomposition

Functional dependencies help identify redundancy of information. To reduce redundancy, a relation can be decomposed in two different relations, which are then reunited with a join. This can not be always be applied because some relations can lead to a lossy decomposition.
We say that a decomposition is a lossless decomposition when there is no loss of information when replacing the relation

$$< R(U), F > \Rightarrow < R_1(X_1), F_1 >, ..., < R_n(X_n), F_n >$$
$$U = X_1 \cup ... \cup X_n$$

Decomposition is not always more efficient, to properly apply a decomposition it must respect this properties:

- The decomposition must be lossless
- redundancies must be eliminated
- The functional dependencies of the original schemas should be preserved

This property are all respected in the schema normal form

Closure of a set of functional dependencies

Given a set of functional dependencies, some are logically implied. We say that a set F of FD logically implies a FD $X \to Y$ if every instance that satisfies F also satisfies $X \to Y$. The set of all FD implied by F is the closure of F, denoted as F^+

To compute the closure of a set, we use the Armstrong's Axioms:

- **Reflexive rule**: if $Y \subseteq X$, then $X \to Y$
- **Augmentation rule**: if $X \to Y$, then $XZ \to YZ$
- **Transitivity rule**: if $X \to Y$ and $Y \to Z$, then $X \to Z$
- **Union rule**: if $X \to Y$ and $X \to Z$, then $X \to YZ$
- **Decomposition rule**:if $X \to YZ$, then $X \to Y$ and $X \to Z$

Closure of a set of attributes

Denoted as $(X)^+$ is the set of all attributes that can be determined by X. This can be used to determine if X si a superkey, a minimal key or nothing at all, if it respects the property $(X)^+ = U.X \rightarrow U$

NB: Some dependencies are defined as trivial when $X \rightarrow Y,\ Y \subseteq X$

8.3 Normal Form

The normal form of a relation is determined when all dependencies $X \rightarrow Y$ in F^+ are listed, with the exception of trivial dependencies (X is superkey of R)

Minimal cover

A simpler version to computing the closure of F is computing the minimal cover. The minimal cover follows this properties:

- $F_{min}^+ = F^+$
- all FD in F_{min} are of form $X \rightarrow A$
- if we modify F_{min}, then $F_{min}^+ \neq F^+$

To compute a minimal cover we follow these steps:

- Normalize each $X \rightarrow ABC$... to $X \rightarrow A$, $X \rightarrow B$,...
- if $XA \rightarrow B$ and $A \in (X)^+$, then is redundant and can be removed
- remove the remaining redundancies

Algoritm for normal form decomposition

- Choose some FD that violates the NF conditions
- Compute $Y = (X)^+/X$ and $Z = U/XY$
- contruct two relation schemas
  $$< R_1(XY), (\Pi_{XY}F^+)_{min} >, < R_2(XZ), (\Pi_{XZ}F^+)_{min} >$$
- if not in normal form, then decompose again

Third normal form

If when for all $\alpha \rightarrow \beta$ in F^+ at least one of the following properties holds:

- $\alpha \rightarrow \beta$ is trivial

- α is superkey of R
- each attribute A in $\beta - \alpha$ is contained in a candidate key for R

then the relation is in third normal form

Cover

G is a cover of F when $F^+ = G^+$. When is composed of only elementary FDs is called canonical cover