0.1 Compotentes

Data Logical view and physical view of data. Logical view is what is seen by the programmers of the application and the physical view is what is seen by the system

People The users are the ones who pay for the system, we need to assume that they know nothing about the system (end user).

Administator is the one who charge of the database.

1 1.4 Procedure

We have large complex data models, but we need a way to populate it with data (need to implement).

We want to reduce the dependencies between programs and data. If we change the data it should have minimal effect on the program (UI excepted). Similarly we want very little change to data when the program is changed. This cannot be eliminated, but reduced.

Three levels (external, conceptual, internal) that each have their own schemas, called the three shema approach

external schema view for users

conceptual schema integrates external schema

internal schema defined physical storage Three schema approach: `https://en.wikipedia.org/wiki/Three_schema_approach`.

When we change the application program only the external schema should change to fit it, maybe the conceptual a bit, but not the internal schema

1.1 Language facilities

Data Definition Language (DDL) - specifies conceptual schema and subschemas and the mappings between them.

Data dictionary, data directory or system catalog - result of compilation of DDL statements in a schema; meta-data.

Data manipulation language (DML) - commands issued in a host program and preprocess or compiler interprets these commands as calls to DBMS. (SQL is default)

Query language a complete language for manipulating data interactively. Should be user friendly but not super powerful (interactive SQL is default)

1.2 Data Models

Value based model requires an attribute that can uniqly identify an object, Object based model does not.

1.3 Database Design Process

Identify functional requirements. From there figure out what data is required for those requirements, called database requirements. Structure this data in a understandable way (make a conceptual schema). This is where we make a entity relationship model. We use entity relationship models in the conceptual and logical (internal) schemas.

1.4 Entity Relationship Model

Invented in 70s and grew in popularity as variations were created, so we have a family of entity relationship models. These can get complicated and hard to read (we use a specific form of notation). An entity is something that has independent existence that is relevant to application. A single value attribute (for example a telephone) has only one value. Multivalue attributes are attributes that might have multiple parts to them (like an address). Some attributes don't belong to anyone and instead describe the relationships between them. Enities in a relationship type model dont have to have a primary key to be used.

- rectangle - entity type
- diamond - relationship type
- circles - attributes

Binary relationship is between two entity types.

1.5 Aggregation

Aggregation is the grouping of entities that have a connected relationship and only one connection outside that group to the rest of the entity map we can redraw the map with those entities grouped into one entity

1.6 Clustering

Store chuncks to data together to make accessing it faster. Try to store a bunch in memory buffer to limit the number of times we need to read.

1.7 Ordered Files

These make searching for specific files much easier (using a binary search since their order is maintained). Whenever you change the main data file all its elements need to be indexed.

B+ tree store the largest index of a block in its parent. There are three techniques, merging, splitting, and distribution. Everything is based on primary key value. The data block is represented by linked pages. Basically store the min and max values of each block in its parent block and you can just run a binary search (note: data must be sorted).

1.8 Relation Scheme

See the big ass example in lecture notes, page 85.

Relational Algebra

Relational tables require all rows to be unique as designated by the primary key (you must have at least one candidate key that is not updated or null). Foreign keys can be used to reference rows in other tables.

```
table_constraint:=
    [CONSTRAINT name]
    {UNIQUE (<column name> + ) |
    PRIMARY KEY (<column name> + ) |
    FOREIGN KEY (<column name> + )
    REFERENCES <table name> [ON {UPDATE |
    DELETE} <effect>]}
<effect> := SET NULL | NO ACTION
    (RESTRICT) | CASCADE | SET DEFAULT
```

Union operation This operation combines two tables if they have the same attributes (all the same columns) and deletes any duplicates. Denoted by $\cup$.

Set Difference This operation returns any elements in one table that are not in the other (tables must have same attributes). R-S results in values that are in R and not in S. Denoted by $-$.

Projection This operation removes or rearranges columns. R(A,B,C) $\rightarrow \pi_{C,A}(R) = R(C,A)$. Denoted by $\pi_{columns}(TABLE)$.

Selection This operations returns rows in a table that satisfy conditions. Denoted by $\sigma_{conditions}$.

Cartesian Product This operation combines two tables by taking their Cartesian product (make all possible combinations of rows). (A, B) X (C D, E F) = (A C D, A E F, B C D, B E F). Denoted by $\times$.

Rename This operation renames columns. Denoted $\rho_{table(columns)}(TABLE)$.

Natural Join This joins two tables based on a shared attribute. Think of a cartesian product but only for rows that have matching attributes. Denoted by $*$.

Binary Relational This operation is a sub operation on natural joins that provides a condition to the join to only include rows that satisfy the condition. This is equivalent to running the set operator on a cartesian product. $R * S_{\theta condition} := \sigma(R \times S)$. Denoted σ.

Intersection This operation returns any rows that are in both tables (tables must have matching attributes). Denoted $\cap$.