**Review:** [**ibelitchka@gmail.com**](mailto:ibelitchka@gmail.com)

**Database:**

* Collection of data that exists over a long period of time
* Examples: phonebook, employee data, university student information, library booklist
* DBMS: DataBase Management System
* DBS = DB + DBMS

**DBMS:**

* Convenient, efficient, secure access and manipulation of large data
* Create, query and modify data
* Controls access to shared data for multiple users (Atomicity, Consistency, Isolation, Durability)
* Advantages: shared data, centralized control, security authorization, reliability, data abstraction
* Three views: internal, conceptual, external

**Architecture:**

* Queries, Transactions, schema creations
* Query processor: queries, modifications
* Storage manager: obtain info requested from data storage, modify info to data storage

**Relational Model:**

* Most widely used model
* Competitor: object oriented model
* Relations = tables
* Database schema: set of table names. Set of attributes for each table
* Ex: Account= {accNum, branchName, amount, customerID}

**Relations:**

* Relation instance: set of tuples
* Users access database through the DBMS
* Instance r of R (R(A)) cannot have infinite tuples, because storage capacity is limited
* **Tuples** are represented in a row
* Relation schema example: Movie (title, year, length)

**Data Independence:**

* Logical: Ability to modify logical schema with little/no affect to rewrite application
* Physical: ability to modify physical schema with no impact on conceptual schema

**Schemas**:

* View, Conceptual, Physical
* Database instance: current content of DB
* Database schema: structure of data
* Ex relation: Students {id, name, department, job, address}
* {} set, () tuple

**SQL:**

* Developed at IBM in 1976
* Replaced all other DB languages
* DDL + DML

**Product:**

* **FROM** clause: couple relations in one query through cartesian product
* **Cartesian** **Product**: set of all pairs (a,b) where a is in R and b is in S -> R x S
* R x S not the same as S x R
* Very expensive for memory
* If one set is empty, we get 0
* SELECT \*, FROM Table: Data manipulation

**Clauses:**

* **SELECT:** selects columns, \* selects everything
* **AS:** renaming attributes: SELECT id AS student\_id
* **FROM:** selects table instance
* **WHERE:** conditioning of table columns
* **GROUP BY:** returns summary of group: SELECT Sid, MAX(grade), FROM Enrolled, GROUP BY Sid;
  + Only those attributes mentioned in the GROUP BY may appear unaggregated in SELECT
* **HAVING**: after GROUP BY, followed by some condition about the group
* **DISTINCT:** select distinct only values, placed after SELECT clause
* **ORDER BY:** orders ASC(default) or DESC, placed after WHERE and FROM (VERY expensive, sort)
* **IN:** retrieve multiple values within WHERE clause, multiple OR operator
* **NOT IN:** negated IN operator
* **LIKE:** after WHERE, checking when value matches pattern

**Aggregate Functions:**

* Used in the SELECT clause, often applied to scalar-valued attribute (column)
* NULL is not included in aggregate functions, has reserved space but it is not a value
* Nulls are counted when grouping, ignored when aggregating
* SELECT COUNT(DISTINCT name): eliminate duplicates, then count (expensive operation)
* **COUNT:** counts number of columns, no NULL values, use DISTINCT for no duplicates
* **MAX**: maximum value of column, no NULL values
* **MIN**: minimum value of column, no NULL values
* **AVG**: average value of column, no NULL values
* **SUM**: sum of columns, no NULL values

**Database Modifications:**

* Do not return tuples, but change state of databases
* Transactions: insert, delete, update
* Insert tuples into relation
* Delete certain tuples from relation
* Update value of certain attribute of certain existing tuples
* **INSERT INTO R(A1,…,An) VALUES (v1,…,vn):** insert match attribute with value, create instance
  + **INSERT INTO R VALUES (v1,…,vn):** ordered values, NULLs are rejected
* **DELETE FROM R WHERE <condition>:** may need WHERE clause
  + **DELETE FROM R; :**  deletes table
  + **DROP TABLE R; :** drop contents of table and its schema
* **UPDATE R SET value = “something” || value WHERE <condition>**

**Database Design:**

* Data model: collection of concepts
* Entity: real world object distinguishable from other objects
* Relationships are association among entities
* Weak entity sets: cannot be a unique entity by itself, uses foreign key (primary key from other)
  + Relation is weak as well (double border)
* Discriminator:

**Tables:**

* **CREATE TABLE** name (list of elements)
  + Principal elements are attributes and their types
  + Declaration of key and constraints may also appear
* **ALTER TABLE** Table\_name **ADD** <column declaration>;
* **ALTER TABLE** Table\_name **DROP COLUMN** <column\_name>;
* **ALTER TABLE** Table\_name **ALTER** column **SET DEFAULT** <value>;
* **DROP TABLE** name;
* **INT, FLOAT, DECIMAL(n, d), NUMERIC, CHAR(n), BIT(B)**
  + **VARCHAR(n):** variant length of char
* **Time:** hh:mm:ss
* **Date:** yyyy-mm-dd (1st m is either 0 or 1)
  + **CREATE TABLE** Days( d DATE);
  + **INSERT INTO** Days **VALUES(‘2012-12-23’)**;

**Keys:**

* Keys are distinct attributes on table to identify unique objects
* Length and size of key is important: smaller the better
* Factors for choosing keys:
  + Total size of attributes
  + Number of attributes
  + Natural choice
* Superkeys: combination of columns that contain uniquely identifiable attributes (primary key)
* Candidate/alternate/minimal keys
* Primary key: candidate key chosen by designer for creation of table
* Foreign key: key from other entity
* Ex: Let R(A,B,C) be relation where B is candidate key, # of superkeys of R?
  + 4: B0, BA, BC, BAC

**Models to remember:**

* **Relation Instance:** table with attributes (columns) and tuples (rows), set of current tuples
* **Relation Schema:** relation name + set of attribute names ( + attribute types)
* **Database Schema:** set of relation schemas D = {R1,…,Rn}
* **Database instance:** collection of relation instances

**Relational Data model:**

* **Input:** E/R Diagram
* **Output:** collection of relational schemas
* **Implicitly:** key attributes of the entity set(s) involved in relationship R
* **Explicitly:** every attribute used explicitly in R
* **Converting isa-**Hierarchies to Relations
  + **Straight E/R style model**
  + **Object-Oriented method**
  + **Nulls method**

**Functional Dependency:**

* Set of attributes X functionally determine a set of attributes Y if value of X determines a unique value of Y
* Key for a relation
* R(A,B,C,D), FD = {AB->C, C->D, D->A} **getting 11 non-trivial FDs**
  + {A}+ = A
  + {B}+ = B
  + {C}+ = C,D,A **{C->A} New Dependency**
  + {D}+ = D,A
  + {A,B}+ = A,B,C,D (candidate key) **{AB->D} New Dependency**
  + {A,C}+ = A,C,D **{AC->D} New Dependency**
  + {A,D}+ = A,D
  + {B,C}+ = B,C,D,A (candidate key) **{BC->A, BC->D} New Dependency**
  + {B,D}+ = B,D,A,C (candidate key) **{BD->A, BD->C} New Dependency**
  + {C,D}+ = C,D,A **{CD->A } New Dependency**
  + {A,B,C}+ = A,B,C,D (candidate key) **{ABC->D} New Dependency**
  + {A,C,D}+ = A,C,D
  + {B,C,D}+ = B,C,D,A (candidate key) **{BCD->A } New Dependency**
  + {A,B,D}+ = A,B,D,C (candidate key) **{ABD->C} New Dependency**
  + {A,B,C,D}+ = A,B,C,D (candidate key)
* Candidate keys: AB, BC, BD, ABCD
* Superkeys but not candidate keys: ABC, ABD, BCD, ABCD
* Finding keys:
  + If attribute never appears on RHS, part of key
  + If attribute never appears on LHS, but appears on RHS, not part of key
* **R = {A,B,C,D}, FD = {AB->C, C->D, C->A}:**
* Implication: If a relation instance r satisfies every FD in a given set F of FDs, then f satisfies F

**Redundancy:**

* piece of information unnecessarily repeated in different tuples in a relation
* main source of problems (storage waste, update, deletion and insertion anomalies)
* R1(SNN, Name) FD: SNN->Name, E2(SNN, Phone) FD: Empty
* Ex: Closures: R(A,B,C) FDs : A->B, B->C
  + Closure of a set X of attributes
  + Closure of a set of FDs
  + {B+} = {B, C}
* F = {A->B, B->C)
* F+ = {A->B, B->C, A->C} -> F+ = F U G (implied)
* Closure: remove redundancies -> remove implications

Schema Refinement:

* Given a set of FDs, How to get a minimal basis?
  + Put every FD in simple form
  + Reduce the LHS of every FD, if needed
  + Make sure the presence of every FD is necessary

**Pattern Matching:**

* **%: any sequence of zero or more characters**

**Truth Values:**

* **1**: True
* **0**: False
* **½**: Unknown

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **P** | **Q** | **OR** | **AND** | **p=q** |
| T | **U** | **T** | **U** | **U** |
| **F** | **T** | **T** | **F** | **F** |
| **F** | **U** | **U** | **F** | **U** |
| **U** | **U** | **U** | **U** | **U** |