**Fall 2017**

**Database Class Notes**

**T.M. Rao**

**Notes-01-Intro**

**What is a Database?**

* Collection of data central to some enterprise
* Essential to operation of enterprise
  + Contains the only record of enterprise activity
* An asset in its own right
  + Historical data can guide enterprise strategy
  + Of interest to other enterprises
* State of database mirrors state of enterprise
  + Database is persistent

**What is a Database Management System?**

* A Database Management System (DBMS) is a program that manages a database:
  + Supports a high-level access language (e.g. SQL).
  + Application describes database accesses using that language.
  + DBMS interprets statements of language to perform requested database access.

**Why use a Database?**

* Databases help people to keep track of things.
* Why not just use a List? Why not use Excel worksheets?

**Example: Lakeview Equipment Rentals:** It rents equipment to other contractors.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Lakeview Equipment Rentals** | | | | | | | | | |
| **Job**  **Loc** | **Contr-**  **actor** | **Phone** | **Eqpmt**  **Type** | **Eqpmt**  **Num** | **Daily**  **Rate**  **($)** | **Start**  **Date** | **End**  **Date** | **Num**  **Days** | **Charge**  **($)** |
| Sea-View Building | KH Serv-  ices | 213-444-1181 | Back-hoe | 10400 | 750 | 6/17/2002 | 6/19/2002 | 3 | 2250 |
| High-Land  Center | Coms-  tock  inc | 232-492-3383 | Back-hoe | 10400 | 750 | 6/24/2002 | 6/24/2002 | 1 | 750 |
| Sea-View  Building | KH  Serv-  ices | 213-444-1181 | Med Crane | 335 | 350 | 6/17/2002 | 7/3/2002 | 17 | 5950 |
| Long Plaza | KH  Serv-  ices | 213-444-1181 | Back-hoe | 10020 | 650 | 7/1/2002 | 7/3/2002 | 3 | 1950 |
| Sea-View  Building | KH  Serv-  ices | 213-444-1181 | Scaffolding |  | 135 | 6/15/2002 |  |  |  |
| High-Land  Center | Coms-  tock  inc | 232-492-3383 | Med Crane | 335 | 400 | 7/1/2002 | 7/8/2002 | 8 | 3200 |
| Village  Square | RB  Partn-  ership | 508-555-3233 | Back-hoe | 10020 | 750 | 7/8/2002 | 7/11/2002 | 4 | 3000 |

**What are the problems with lists?**

* What if the contractor KH Services changes phone number to 213-444-9988?
  + Inconsistency
  + Error-prone and time-consuming
* What if we delete a record from the list?
  + RB Partnership decides not to rent the backhoe! Loss of information
* What if we have some, but not all of the data?
  + University Swaging has a phone # 206-555-8989 and it is working on a project called Center Street Bridge Renewal. But they have not yet rented anything from us. How do we store this data?
* Inconsistencies: Simple keying mistakes:
  + Backhoe 10020: In row 4, the rate is 650, in row 7 it is 750. Is this a mistake?

**More Problems because of shared data:**

* Everyone should have access to contractor names and phone numbers
* Accounting does not want anybody outside to see accounting info.
* Most departments want to share some of its data but not all the data**.**

**Better to use a database:** A relational database consists of tables. For this problem we can design tables like this:

**JOB\_LOCATIONS**

|  |  |
| --- | --- |
| **ID** | **JobName** |
| 1 | Sea View Building |
| 2 | Highland Center |
| 3 | Long Plaza |
| 4 | Village Square |

**CONTRACTORS**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **Contractor** | **Phone** | **Street1** | **Street2** | **City** | **State** | **Zip** |
| 1 | KH Services | 213-444-1181 | 111 Pine |  | New York | NY | 12345 |
| 2 | Comstock Inc | 232-492-3383 | 1200 Comstock |  | New York | NY | 12345 |
| 3 | RB Partnership | 508-555-3233 | 1234 Elm |  | Highlands | CA | 94595 |

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Equipment Type** | **Equipment Number** | **Daily Rate** |
| 1 | Backhoe | 10400 | 750.00 |
| 2 | Backhoe | 10020 | 650.00 |
| 3 | Medium Crane | 335 | 350 |
| 4 | Scaffolding |  | 135 |

**EQUIPMENT**

**RENTAL**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **JobID** | **Contractor**  **ID** | **Equipment**  **ID** | **Start**  **Date** | **End**  **Date** |
| 1 | 1 | 1 | 1 | 6/17/2002 | 6/19/2002 |
| 2 | 2 | 2 | 1 | 6/24/2002 | 6/24/2002 |
| 3 | 1 | 1 | 2 | 6/17/2002 | 7/3/2002 |
| 4 | 3 | 1 | 3 | 7/1/2002 | 7/3/2002 |
| 5 | 1 | 1 | 4 | 6/15/2002 |  |
| 6 | 2 | 2 | 2 | 7/1/2002 | 7/8/2002 |
| 7 | 4 | 3 | 3 | 7/8/2002 | 7/8/2002 |

**Does this design solve the problems we mentioned before?**

* What if the contractor KH Services changes phone number to 213-444-9988?
  + We just update the **contractors** table
* What if we delete a record from the list?
  + RB Partnership decides not to rent the backhoe! We delete one line from rentals table. We still have info about them in the **contractors** table.
* What if we have some, but not all of the data?
  + University Swaging has a phone # 206-555-8989 and it is working on a project called Center Street Bridge Renewal. But they have not yet rented anything from us. How do we store this data? We add this to the **contractors** table.
* Inconsistencies: Simple keying mistakes:
  + Backhoe 10020: In row 4, the rate is 650, in row 7 it is 750. Is this a mistake? – We see that there are two types of backhoes. So, there is no mistake.

**Components of a Database System**

* Users
* Application Programs
* DBMS
* The Database

USER

Database

Application

DBMS

**DB**

**Users**

* Employ database to perform their jobs
* Input new data, modify existing data, delete data.
* They use forms, queries and reports

**Functions of a Database Application Program**

* Create and Process Forms
* Create and Process Queries
* Create and Process Reports
* Update the database (via DBMS) based on program logic

**Functions of the DBMS**

* Create the database itself
* Create tables
* Create supporting structures (such as indexes)
* Support read/write/update data in database
* Maintain Database Structures (Functional Dependencies, etc.)
* Control Concurrency
* Provide Security
* Perform backup and recovery

**Database**:

* In the most general case a database is a self-describing collection of related records.
* Specifically, a **relational database** is a self-describing collection of related tables.
* By **self-describing** we mean that the description of the structure of the data is contained within the database itself.

**Components of a database:**

* User data
* Metadata
* Indexes and related structures
* Stored procedures
* Triggers
* Application Metadata

# User Data

* User data is represented as relations –or- tables.
* Rows of the table are “records”
* Columns are “fields” or “attributes”
* Relations must be structured properly

**Metadata**

* “A description of the structure of the database”
* System Tables store metadata

**Indexes**

* Sometimes called “overhead data” improves the performance and accessibility.

**Stored Procedures**

* A program stored within the database that does a specific job when called upon.

**Triggers**

* A trigger is a special procedure that gets automatically invoked when some event takes place

**Application Metadata**

* Stores the format of forms, queries, reports and other application components.

**Naming Conventions**

* There is no ‘Standard’. We shall use the following conventions:
* Names of tables are in all capitals: STUDENT, COURSE\_OFFERING, GRADE, etc.
* Names of fields within a table: Capitalized. If there are multiple words, each word is capitalized (Camel Notation): StudentName, StudentNumber, CourseNumber, etc.

**What is a Transaction?**

* When an event in the real world changes the state of the enterprise, a transaction is executed to cause the corresponding change in the database state
  + With an on-line database, the event causes the transaction to be executed in real time
* A transaction is an application program with special properties - discussed later - to guarantee it maintains database correctness

**What is a Transaction Processing System?**

* Transaction execution is controlled by a TP monitor
  + Creates the abstraction of a transaction, analogous to the way an operating system creates the abstraction of a process
  + TP monitor and DBMS together guarantee the special properties of transactions
* A Transaction Processing System consists of TP monitor, databases, and transactions

**Transaction Processing System**

**System Requirements**

* High Availability: on-line => must be operational while enterprise is functioning
* High Reliability: correctly tracks state, does not lose data, controlled concurrency
* High Throughput: many users => many transactions/sec
* Low Response Time: on-line => users are waiting
* Long Lifetime: complex systems are not easily replaced
  + Must be designed so they can be easily extended as the needs of the enterprise change
* Security: sensitive information must be carefully protected since system is accessible to many users
  + Authentication, authorization, encryption

**Roles in Design, Implementation, and Maintenance of a TPS**

* System Analyst - specifies system using input from customer; provides complete description of functionality from customer’s and user’s point of view
* Database Designer - specifies structure of data that will be stored in database
* Application Programmer - implements application programs (transactions) that access data and support enterprise rules
* Database Administrator - maintains database once system is operational: space allocation, performance optimization, database security
* System Administrator - maintains transaction processing system: monitors interconnection of HW and SW modules, deals with failures and congestion

**Transactions are different from ordinary programs**

* Many enterprises use databases to store information about their state
  + *E.g*., balances of all depositors
* The occurrence of a real-world event that changes the enterprise state requires the execution of a program that changes the database state in a corresponding way
  + *E.g*., balance must be updated when you deposit
* A *transaction* is a program that accesses the database in response to real-world events
* Transactions are not just ordinary programs
* Additional requirements are placed on transactions (and particularly their execution environment) that go beyond the requirements placed on ordinary programs.
  + Atomicity
  + Consistency
  + Isolation
  + Durability

**Integrity Constraints**

* Rules of the enterprise generally limit the occurrence of certain real-world events.
  + Student cannot register for a course if current number of registrants >= maximum allowed
* Correspondingly, allowable database states are restricted.
  + *cur\_reg <= max\_reg*
* These limitations are expressed as *integrity constraints,* which are assertions that must be satisfied by the database state.

**Consistency**

* Transaction designer must ensure that

IF the database is in a state that satisfies all integrity constraints when execution of a transaction is started

THEN when the transaction completes:

* + - All integrity constraints are once again satisfied
    - New database state satisfies specifications of transaction

**Atomicity**

* A real-world event either happens or does not happen.
  + Student either registers or does not register.
* Similarly, the system must ensure that either the transaction runs to completion (*commits*) or, if it does not complete, it has no effect at all (*aborts*).
  + This is not true of ordinary programs. A hardware or software failure could leave files partially updated.

**Durability**

* The system must ensure that once a transaction commits its effect on the database state is not lost in spite of subsequent failures.
  + Not true of ordinary systems. For example, a media failure after a program terminates could cause the file system to be restored to a state that preceded the execution of the program.

**Isolation**

* Deals with the execution of multiple transactions concurrently.
* If the initial database state is consistent and accurately reflects the real-world state, then the *serial* (one after another) execution of a set of consistent transactions preserves consistency.
* But serial execution is *inadequate* from a performance perspective.

**ACID Properties**

* The transaction monitor is responsible for ensuring atomicity, durability, and (the requested level of) isolation.
  + Hence it provides the abstraction of failure-free, non-concurrent environment, greatly simplifying the task of the transaction designer.
* The transaction designer is responsible for ensuring the consistency of each transaction, but doesn’t need to worry about concurrency and system failures.

**OLTP vs OLAP:** These are different types of databases. An OLTP database is used/updated every day whereas OLAP is for historical data.

* On-line Transaction Processing (OLTP)
  + Day-to-day handling of transactions that result from enterprise operation
  + Maintains correspondence between database state and enterprise state
* On-line Analytic Processing (OLAP)
  + Analysis of information in a database for the purpose of making management decisions

**OLAP**

* Analyzes historical data (terabytes) using complex queries
* Due to volume of data and complexity of queries, OLAP often uses a data warehouse
* Data Warehouse - (offline) repository of historical data generated from OLTP or other sources
* Data Mining - use of warehouse data to *discover* relationships that might influence enterprise strategy

**Examples - Supermarket**

* OLTP
  + Event occurred: Customer bought 3 cans of soup and 1 box of crackers; update database to reflect that event
* OLAP
  + Last winter in all stores in northeast, how many customers bought soup and crackers together?
* Data Mining
  + Are there any interesting combinations of foods that customers frequently bought together?