**INFO 6210 Section 04 Hospital Management System**

**Name: Leandra Menezes**

**NUID: 001621189**

**SET 1:**

**CREATE DATABASE**

CREATE SCHEMA IF NOT EXISTS `hospital\_mgmt\_sys` DEFAULT CHARACTER SET utf8;

**USE**

USE hospital\_mgmt\_sys;

**DROP**

DROP SCHEMA IF EXISTS `hospital\_mgmt\_sys` ;

**CREATE TABLE:**

CREATE TABLE IF NOT EXISTS `patient` (

`patient\_id` VARCHAR(15) NOT NULL COMMENT '',

`patient\_name` VARCHAR(100) NOT NULL COMMENT '',

`patient\_dob` DATE NOT NULL COMMENT '',

`patient\_gender` VARCHAR(10) NOT NULL COMMENT '',

`patient\_street` VARCHAR(100) NOT NULL COMMENT '',

`patient\_city` VARCHAR(100) NOT NULL COMMENT '',

`patient\_zip` INT(11) NOT NULL COMMENT '',

`patient\_apt\_no` VARCHAR(15) NULL COMMENT '',

`patient\_cont\_no` VARCHAR(100) NOT NULL COMMENT '',

PRIMARY KEY (`patient\_id`) COMMENT '',

UNIQUE INDEX `Patientid\_UNIQUE` (`patient\_id` ASC) COMMENT '')

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

**INSERT INTO**

INSERT INTO `hospital\_mgmt\_sys`.`patient` (`patient\_id`, `patient\_name`, `patient\_dob`, `patient\_gender`, `patient\_street`, `patient\_city`, `patient\_zip`, `patient\_apt\_no`, `patient\_cont\_no`) VALUES ('p1', 'Kelsey Wise', '1960/02/21', 'Male', '728-1625 Mauris, St.', 'Boston', '41609', '41', '61302');

**Set 2:**

**CREATE TABLE**

DROP TABLE IF EXISTS `accounts`;

CREATE TABLE IF NOT EXISTS `accounts` (

`accounts\_case\_no` VARCHAR(15) NOT NULL COMMENT '',

`patient\_id` VARCHAR(15) NOT NULL COMMENT '',

`accounts\_bed\_id` INT(11) NULL COMMENT '',

`accounts\_admitted\_date` DATE NULL COMMENT '',

`accounts\_admitted\_discharge\_date` DATE NULL COMMENT '',

`accounts\_illness\_detected` VARCHAR(40) NULL COMMENT '',

PRIMARY KEY (`accounts\_case\_no `) COMMENT '',

CONSTRAINT `fk\_bed\_pat`

FOREIGN KEY (`patient\_id `)

REFERENCES `PATIENT` (`patient\_id`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

**INSERT INTO**

INSERT INTO `hospital\_mgmt\_sys`.`testlog` (`testlog\_id`, `nurse\_id`, `doctor\_id`, `patient\_id`, `test\_type`, `test\_date`, `test\_result\_date`) VALUES ('1', 'E21', 'E2', 'p7', 'MRI', '2015-02-01', '2015-02-04');

**CREATE INDEX**

CREATE INDEX EIndex

ON employee (employee\_name);

CREATE INDEX PIndex

ON patient (patient\_name);

**SET 3:**

**SELECT:**

select \* from employee;

**SELECT FROM GROUP BY:**

SELECT employee\_qualification, COUNT(\*)

FROM employee GROUP BY employee\_qualification;

**SELECT ORDER BY**

SELECT \* FROM patient

ORDER BY patient\_city DESC;

**SELECT GROUP BY HAVING**

SELECT employee\_name, MAX(employee\_salary) FROM employee

GROUP BY employee\_qualification HAVING MAX(employee\_salary) > 50000;

**LIMIT**

SELECT testlog\_id, patient\_id, test\_type

FROM testlog

ORDER BY test\_type ASC

LIMIT 7;

**SET 4:**

**FUNCTIONS**

SELECT patient\_name, patient\_city

FROM patient

WHERE patient\_city LIKE 'B%';

SELECT employee\_name, employee\_salary, IF(employee\_salary>= 1000000, 'High Salary', 'Low Salary') AS Salary\_level

FROM employee;

SELECT accounts\_case\_no,

CONCAT('Admit Date(',

CAST(accounts\_admit\_date AS date),

',',

CAST(accounts\_admit\_discharge\_date AS date),

')') patientDate

FROM accounts;

SELECT patient\_name, SUBSTRING(patient\_name,2,5) FROM patient

WHERE patient\_gender='Male';

SELECT employee\_name, SQRT(employee\_salary)

FROM employee;

SELECT testlog\_id, test\_date, MONTH(test\_date)

FROM testlog

WHERE MONTH(test\_date)>02;

SELECT employee\_name, employee\_salary FROM employee

WHERE employee\_salary>(SELECT AVG(employee\_salary) FROM employee);

SELECT test\_type, COUNT(\*) FROM testlog

WHERE test\_type = 'XRAY';

SELECT MIN(employee\_salary) least, MAX(employee\_salary) max FROM employee;

SELECT patient\_name,patient\_city,

REVERSE(patient\_city)

FROM patient

WHERE patient\_city='Chicago';

**SET 5:**

**INNER JOIN**

SELECT patient.patient\_name, accounts.accounts\_case\_no

FROM patient

INNER JOIN accounts

ON patient.patient\_id=accounts.patient\_id

ORDER BY patient.patient\_name;

**LEFT JOIN**

SELECT testlog.testlog\_id, testlog.test\_type, patient.patient\_id

FROM testlog

LEFT JOIN patient

ON testlog.patient\_id=patient.patient\_id

ORDER BY testlog.test\_type;

**RIGHT JOIN**

SELECT patient.patient\_name, accounts.accounts\_bed\_id, account\_illness\_detected

FROM patient

RIGHT JOIN accounts

ON patient.patient\_id=accounts.patient\_id

ORDER BY accounts.accounts\_case\_no;

**FULL JOIN**

(SELECT patient.patient\_name, patient.patient\_gender, doctor\_examines\_patient.doctor\_id

FROM patient

LEFT JOIN doctor\_examines\_patient

ON patient.patient\_id=doctor\_examines\_patient.patient\_id)

UNION

(SELECT patient.patient\_name, patient.patient\_gender, doctor\_examines\_patient.doctor\_id

FROM patient

RIGHT JOIN doctor\_examines\_patient

ON patient.patient\_id=doctor\_examines\_patient.patient\_id);

SELECT patient\_name, accounts\_admit\_date, accounts\_admit\_discharge\_date

FROM patient, accounts;

**SET 6:**

**SUBQUERIES**

select patient\_name from patient where patient\_id IN (Select patient\_id from accounts where account\_illness\_detected = 'myeloma');

select employee\_name from employee where employee\_id IN(Select doctor\_id from testlog where patient\_id

IN(Select patient\_id from accounts where account\_illness\_detected = 'thyroid' ));

**SET 7:**

**USER PRIVILEGES**

CREATE USER SYS\_ADMIN;

CREATE USER DEVELOPER;

CREATE USER SYS\_TESTER;

CREATE USER USER;

GRANT ALL ON hospital\_mgmt\_sys.\* TO SYS\_ADMIN;

GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, TRIGGER, INDEX ON hospital\_mgmt\_sys.\* TO DEVELOPER;

GRANT SELECT, INSERT, UPDATE, DELETE ON hospital\_mgmt\_sys.\* TO SYS\_TESTER;

GRANT SELECT ON hospital\_mgmt\_sys.\* TO USER;

REVOKE ALL ON hospital\_mgmt\_sys.\* FROM SYS\_ADMIN;

REVOKE SELECT, INSERT, UPDATE, DELETE, CREATE, ALTER, TRIGGER, INDEX ON hospital\_mgmt\_sys.\* FROM DEVELOPER;

REVOKE SELECT, INSERT, UPDATE, DELETE ON hospital\_mgmt\_sys.\* FROM SYS\_TESTER;

REVOKE SELECT ON hospital\_mgmt\_sys.\* FROM USER;

DROP USER SYS\_ADMIN;

DROP USER DEVELOPER;

DROP USER SYS\_TESTER;

DROP USER USER;

**SET 8:**

**STORED PROCEDURE**

DROP PROCEDURE IF EXISTS EMPLOYEE\_LUCKY\_DRAW;

#PICK A LUCKY DRAW FOR EMPLOYEES

DELIMITER $$

CREATE PROCEDURE EMPLOYEE\_LUCKY\_DRAW ()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE LUCKY VARCHAR(100) DEFAULT "";

DECLARE CAND VARCHAR(100) DEFAULT "";

DECLARE CUR1 CURSOR FOR SELECT employee\_id FROM employee;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

SELECT (1+FLOOR(40\*RAND())) INTO LUCKY FROM DUAL;

OPEN CUR1;

LUCK\_TEST : LOOP

FETCH CUR1 INTO CAND;

IF done THEN

LEAVE LUCK\_TEST;

END IF;

IF CAND LIKE CONCAT("%e",LUCKY,"%") THEN

SELECT CONCAT("e",LUCKY) as lucky\_candidate FROM DUAL;

LEAVE LUCK\_TEST;

END IF;

END LOOP LUCK\_TEST;

CLOSE CUR1;

END$$

DELIMITER ;

CALL EMPLOYEE\_LUCKY\_DRAW();

**STORED PROCEDURE 2:**

DROP PROCEDURE IF EXISTS SAL\_INCR\_NIGHT\_ALLOWANCE;

DELIMITER $$

CREATE PROCEDURE SAL\_INCR\_NIGHT\_ALLOWANCE ()

BEGIN

DECLARE done INT DEFAULT FALSE;

DECLARE EMP\_ID VARCHAR(100);

DECLARE SAL INT DEFAULT 0;

DECLARE SFT VARCHAR(100);

DECLARE CUR1 CURSOR FOR SELECT employee\_id,employee\_shift,employee\_salary FROM employee;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

OPEN CUR1;

SAL\_ADJUST : LOOP

FETCH CUR1 INTO EMP\_ID,SFT,SAL;

IF done THEN

LEAVE SAL\_ADJUST;

END IF;

IF SFT = "Night" THEN

UPDATE employee SET employee\_salary = SAL + 200 WHERE employee\_id = EMP\_ID;

END IF;

END LOOP SAL\_ADJUST;

CLOSE CUR1;

END$$

DELIMITER ;

CALL SAL\_INCR\_NIGHT\_ALLOWANCE();

SELECT \* FROM EMPLOYEE WHERE employee\_shift = "night";

**SET 9:**

**TRANSACTIONS**

START TRANSACTION;

LOCK TABLE employee WRITE;

SAVEPOINT SVPNT2;

UPDATE employee SET employee\_name = "Shruti" WHERE employee\_id = "E1";

UPDATE employee SET employee\_name = "Jasmine" WHERE employee\_id = "E11";

UPDATE employee SET employee\_name = "Kenneth" WHERE employee\_id = "E12";

ROLLBACK TO SVPNT2;

UNLOCK TABLE;

**SET 10:**

**TRIGGER ON INSERT**

DROP TRIGGER IF EXISTS trg\_employee\_ins;

DELIMITER &&

CREATE TRIGGER trg\_employee\_ins BEFORE INSERT ON employee

FOR EACH ROW

BEGIN

DECLARE msg VARCHAR(100);

IF NEW.employee\_qualification LIKE '%MBBS%' OR '%DO%' OR '%MM%' OR '%MS%' AND NEW.employee\_salary < 1000000 THEN

SET msg = concat('My Trigger: Doctors should get atleast $1000000');

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = msg;

ELSEIF NEW.employee\_qualification LIKE '%BSN%' OR '%MSN%' AND NEW.employee\_salary < 5000 THEN

SET msg = concat('My Trigger: Nurse should get atleast $5000');

SIGNAL SQLSTATE '46000' SET MESSAGE\_TEXT = msg;

END IF;

END

&&

DELIMITER &&

insert into employee values ('E50', 'Francis', 4000, '2015-03-14', 'BSN', 'Morning', 56789);

**TRIGGER ON UPDATE:**

DROP TRIGGER IF EXISTS trg\_admit\_upd;

DELIMITER &&

CREATE TRIGGER trg\_admit\_upd BEFORE UPDATE ON accounts

FOR EACH ROW

BEGIN

DECLARE msg VARCHAR(100);

IF old.accounts\_admit\_date > new.accounts\_admit\_discharge\_date then

set msg = concat('My Trigger: Admitted date cannot be greater than discharge date : ', cast(new.accounts\_admit\_discharge\_date as char));

signal sqlstate '30000' set message\_text = msg;

end if;

end

&&

delimiter ;

select \* from accounts;

select \* from patient;

update accounts

set accounts\_admit\_discharge\_date = '2015-01-04'

where patient\_id = 'p11';

**TRIGGER ON DELETE:**

create table employee\_backup like employee;

alter table employee\_backup add column resign\_date date;

drop trigger if exists trg\_employee\_del;

DELIMITER &&

CREATE TRIGGER trg\_employee\_del BEFORE DELETE ON employee

FOR EACH ROW

BEGIN

DECLARE dt DATE;

SET dt = curdate();

INSERT INTO employee\_backup VALUE(old.employee\_id,old.employee\_name,old.employee\_salary, old.employee\_join\_date,old.employee\_qualification,old.employee\_shift,employee\_contact\_no,dt);

END &&

DELIMITER ;

select \* from employee;

delete from employee where employee\_id = 'E1';

select \* from employee\_backup;

commit;

**SET 11:**

**VIEW**

CREATE OR REPLACE VIEW Employee\_List AS

SELECT employee\_id,employee\_name, employee\_salary

FROM employee

WHERE employee\_salary> 7000;

SELECT \* from Employee\_List;

CREATE OR REPLACE VIEW Patient\_Account AS

SELECT patient.patient\_name, accounts.accounts\_bed\_id, account\_illness\_detected, accounts\_admit\_date, accounts\_admit\_discharge\_date

FROM patient

RIGHT JOIN accounts

ON patient.patient\_id=accounts.patient\_id

ORDER BY accounts.accounts\_case\_no;

SELECT \* from Patient\_Account;

CREATE OR REPLACE VIEW Employee\_Illness AS

SELECT employee\_name FROM employee WHERE employee\_id IN(SELECT doctor\_id FROM testlog WHERE patient\_id

IN(SELECT patient\_id FROM accounts WHERE account\_illness\_detected = 'thyroid' ));

SELECT \* from Employee\_Illness;

**Create a backup plan of your choice, state your assumptions/constraints and explain your reasoning. Print your selection on a sample monthly calendar (say 30 day) with FB for Full Backup and IB for Incremental Backup**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Sunday** | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** |
|  |  | 1  Incremental Backup from last saturday | 2  Incremental  Backup from last saturday | 3  Incremental  Backup from last wednesday | 4  Incremental Backup from last wednesday | 5  Full Backup |
| 6  Incremental Backup from last saturday | 7  Incremental Backup from last saturday | 8  Incremental Backup from last saturday | 9  Incremental Backup from last saturday | 10  Incremental  Backup from last wednesday | 11  Incremental Backup from last wednesday | 12  Full Backup |
| 13  Incremental Backup from last saturday | 14  Incremental Backup from last saturday | 15  Incremental Backup from last saturday | 16  Incremental Backup from last saturday | 17  Incremental Backup from last wednesday | 18  Incremental Backup from last wednesday | 19  Full Backup |
| 20  Incremental Backup from last saturday | 21  Incremental Backup from last saturday | 22  Incremental Backup from last saturday | 23  Incremental Backup from last saturday | 24  Incremental Backup from last wednesday | 25  Incremental Backup from last wednesday | 26  Full Backup |
| 27  Incremental Backup from last saturday | 28  Incremental Backup from last saturday | 29  Incremental Backup from last saturday | 30  Incremental Backup from last saturday | 31  Incremental Backup from last wednesday |  |  |

**Assumptions:**

* A full backup takes around 12 hours
* Incremental backup takes 2 hours
* Patients in the night as lesser than patients in the morning

**Backup idea:**

An incremental backup on Sunday, Monday, Tuesday and Wednesday takes around 2 hours so it can save changes from the previous 4 days. All these backups will record the changes made in the database since the last full backup on Saturday. In order to retrieve one of these backups the only thing that has to be done is apply previous full backup and then one incremental backup.

On Thursday night and Friday night an incremental backup since Wednesday is done. So if there is a need to restore data, restore data from last backup then the Wednesday night backup and then this backup.

**4) What are Data Warehousing (DW) and Business Intelligence? Describe its components and its uses. How does DW the same or different with respect to Business Intelligence? Define and explain ETL operations.**

Business Intelligence (BI) is the tools and systems that play an important role in the planning and growing process of the corporation. They allow a company to gather, store,access and analyze corporate data to aid in decision making process.

A data warehouse (DW) on the other hand is a relational database especially designed for query and analysis rather than for transaction processing. It contains historical data derived from transaction data, but this data could even come from other sources. It separates analysis workload from transaction workload and enables an organization to consolidate data from several sources.

**Datawarehousing components:**

**Data sources:** Data are sourced from operational systems and possibly from external data sources.

**Data extraction.** Data is extracted using custom-written or commercial software called ETL.

**Data loading.** In this stage data is loaded into a staging area. This data is then transformed and cleansed. Once data is transformed and cleansed it is ready to be loaded in the data warehouse.

**Comprehensive database.** This is the enterprise data warehouse that supports decision analysis by providing relevant summarized and detailed information.

**Metadata.** Metadata is data that describes other data. They are maintained for access by IT personnel and users. Metadata include rules for organizing data summaries that are easy to index and search.

**Middleware tools.** Middleware tools as the name suggests allows access to the data warehouse from a wide variety of front end applications.

**Business Intelligence Components:**

**OLAP (Online Analytical Processing):** OLAP refers to the computing process that enables business users to slice and dice their way through data using tools that allow the user to easily and selectively view data from different points of view.

**Advanced Analytics or Corporate Performance Management (CPM):** can also be referred as data mining, forecasting or predictive analytics. It takes advantage of statistical analysis techniques to predict or provide certainty measures on facts.

**Real-time BI:** Real time means near to zero latency and allows for the real time distribution of metrics through email, messaging systems and/or interactive displays.

**Data Warehousing:** Data warehousing focusses on capturing of data from various sources useful analysis and access. It helps in business as it helps business leaders to explore through subsets of data and examine interrelated components that can help drive business.

**Data Sources:** Data source as the name suggests is a source of data. The data sources can be relational database, file, any other computer software, tables, spreadsheets, or unstructured information, such as plaintext files or pictures and other multimedia information. The main purpose of data source is to gather the technical information needed to access the data from different sources.

**Data warehouse and BI are different in the following ways:**

Data warehouse is the technique of storing data and creating information through data marts. Data marts are segments of data or information that are combined together to provide useful information to those segments. Data warehouse does not need Business Intelligence to work. Reports can be easily generated from reporting tools.

Business Intelligence are a set of tools and methods that take raw data and transform it into useful information. This information can help make business decisions and recommendations through the use of statistical analysis tools and data mining tools.

**ETL Process:**

**Extraction Transformation and loading**

**Extraction:** Focuses on retrieving the data from the storage source. Common data source structures are relational databases and pure data files. Other sources of data could include non-relational database patterns or other data structures like virtual storage access method (VSAM), indexed sequential access method (ISAM). Data sources can even include external sources such as data coming from the Internet.

The transform phase is the second step of ETL process. In this stage a series of rules and operations are performed to retrieve pure data from the source to deliver the data for manipulation at the other end. Some data source may need little processing or may not need at all. Sometimes a combination of more than one transformation is done on the target database to meet the business requirements.

The loading stage is the last process of ETL. It aims at sending data to the receiving end. Either this process may be very difficult or easy depending on the needs of the application. Sometimes the receiving end may replace old data with cumulative data. Updating of data in the data storage is usually done on a periodic basis.

References: <http://www.webopedia.com/TERM/B/Business_Intelligence.html>

<https://docs.oracle.com/cd/B10500_01/server.920/a96520/concept.htm>

<http://www.jatit.org/volumes/research-papers/Vol9No1/9Vol9No1.pdf>

<http://www.villanovau.com/resources/bi/overview-of-business-intelligence-bi-components/#.VnH9qEorJD9>

<http://datawarehouse.ittoolbox.com/documents/data-warehousing-vs-business-intelligence-18494>

<http://www.dataintegration.info/etl>

**5) What is unstructured database (like Hadoop, Casandra, No-SQL) and how is it different from the traditional DB and its application?**

Unstructured data is data that does not traditionally be in row and column structure. It is typically opposite of structured data.

**Hadoop:** is an open-source framework that allows easy storing and processing big data in a distributed environment across clusters of computers using simple programming models. Hadoops allows business to quickly gain insight from structured and unstructured database.

**Cassandra:** is distributed database for managing large amounts of structured data across many commodity servers. It is not just limited to managing data but also providing availability of service and no single point of failure.  Cassandra offers benefits that no other relational database or No SQL can provide: continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers and cloud availability zones.

**No-SQL:** A NoSQL database is a non-relational and largely distributed database system. It enables rapid, ad-hoc organization and analysis of extremely high-volume, disparate data types. They are also referred to as cloud databases, non-relational databases, Big Data databases etc.

**How is it different from traditional DB:**

* SQL databases are Relational Databases (RDBMS) whereas NoSQL database are non-relational or distributed database.
* No SQL supports semi-structured data and volatile data
* SQL databases are table based databases whereas NoSQL databases are document based, key-value pairs, graph databases or wide-column stores.
* SQL databases are vertically scalable whereas the NoSQL databases are horizontally scalable.
* SQL databases uses SQL (structured query language) for defining and manipulating the data, which is very powerful whereas in NoSQL database queries are focused on collection of documents.
* No-SQL supports Big data in volumes of Terra Bytes & Peta Bytes
* SQL databases are not best fit for hierarchical data storage but NoSQL database fits better for the hierarchical data storage. The reason being that No SQL follows key-value pair way of storing data.
* With Relational Database Management Systems, built-in clustering is difficult due to the ACID properties of transactions whereas clustering it is easier in No SQL.
* SQL databases emphasizes on ACID properties ( Atomicity, Consistency, Isolation and Durability) whereas the NoSQL database follow the Brewers CAP theorem ( Consistency, Availability and Partition tolerance )
* SQL databases are best fit for heavy duty transactional type applications whereas you can use NoSQL for transactions purpose but it is still not comparable and sable enough for complex transactional applications.

<http://hortonworks.com/hadoop/>

<http://www.thegeekstuff.com/2014/01/sql-vs-nosql-db/>

**6) What is the mechanism behind secure delete in DB? Describe the steps/method to securely delete data from MySQL and/or MS SQL SERVER DB? How is the process any different from a DELETE call in SQL?**

In MYSQL databases if we fire a delete query on the database the data gets deleted but it is not completely destroyed. The reason behind this is that deletion is logical data not deleted. Deletion is insecure. With simple analysis of databases and basic knowledge on database internals, we can reconstruct the deleted data item. Retrieval of data can take place using forensic recovery tools which can be clearly very risky for the company.

In order to make sure that the data is completely deleted from the system you can use the following mechanism:

1. Overwrite data with zeroes: First update the entire data with empty data of the same length. After that delete the rows and add few rows with random data. After entering the data delete these row. Drop the database and run a shrink command, re-index or compact command.

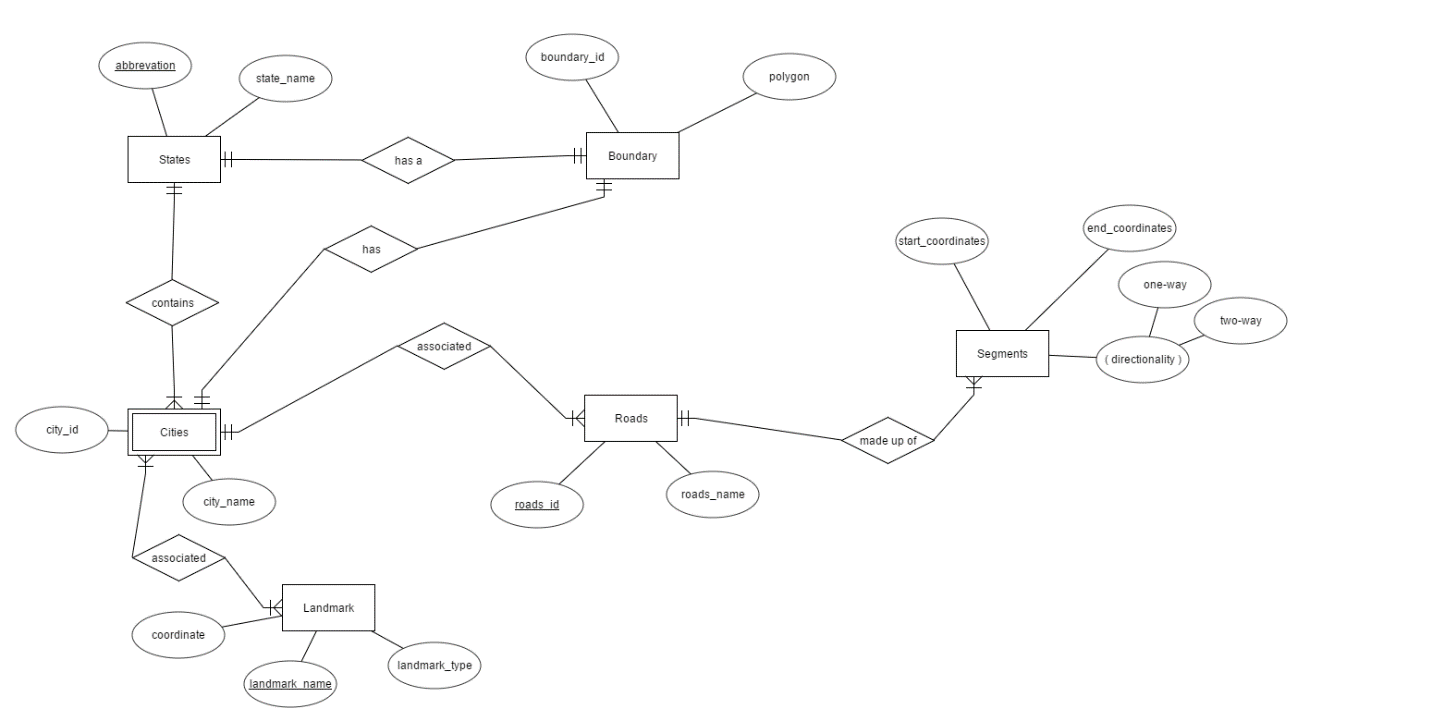
2. Store data in encrypted form and disposing the keys: In this stage encrypt the data and then delete it by throwing out the keys used in encryption algorithms. Reason behind this is now if the data is recovered the decryption of the data won’t be possible without the use of keys.

References: <http://db.csail.mit.edu/nedbday08/slides/miklau.pdf>

<http://www.infosun.fim.uni-passau.de/publications/docs/GSK13.pdf>

**7)** **Suppose we are commissioned to design a schema for a new map and direction-finding site, Mondo Maps. Like MapQuest and Google Maps, our site needs to display route and map information. Underneath it lies a database of cities, states, roads, and landmarks, in a two-dimensional plane (with longitude and latitude specifying a coordinate). • States have abbreviations (unique) and names, and each state has a unique boundary. • Cities are unique within states and have names, and each city has a single boundary. • A boundary corresponds to a city or state, and it has an ID and a polygon. (Assume there is a special polygon data type.) • Roads have IDs and names, and are made up of multiple segments. Assume that a road is associated with a single city. • A road segment has a start and an end coordinates, as well as directionality (one-way or two-way). • A landmark has a single coordinate, a name, and a type. Assume landmarks are associated with cities, and that landmark names are globally unique.**

**7a) Draw an ER diagram for this domain. Include participation constraints. You may need to add relationship sets that weren’t explicitly specified. Many options were acceptable here. However, it is important to note that cities are weak entity sets as defined.**

****

**8) Provide definition and example of the following normal forms: 1NF, 2NF, 3NF and 3.5NF. Give two examples of each, a) when would you want to use normalization; b) when not to use normalization**

A relation is in the first normal form if the values that are in the domain of each attribute of relation are atomic. 1NF has three basic rules:

* Remove identical columns from the same table.
* Identify every row with a unique column or set of columns (primary key)
* Develop separate table for every group of related values.

For eg:

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp\_id** | **Emp\_name** | **Emp\_address** | **Emp\_mobile** |
| 101 | John | Mumbai | 96452120 |
| 102 | Vita | Boston | 78942351  85642354 |
| 103 | Allen | Chicago | 87246590 |
| 104 | Neola | New York | 12496354 |

The table is not in 1NF as each attribute in the table must have unique values.

Hence the 1NF form is as below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp\_id** | **Emp\_name** | **Emp\_address** | **Emp\_mobile** |
| 101 | John | Mumbai | 96452120 |
| 102 | Vita | Boston | 78942351 |
| 102 | Vita | Boston | 85642354 |
| 103 | Allen | Chicago | 87246590 |
| 104 | Neola | New York | 12496354 |

**2NF:**

In 2NF the following rules are important:

* No partial dependency of any column on primary key
* Meet all the requirement of the first normal form
* For a table that has a concatenated primary key, each column in the table that is not part of the primary key must depend upon the entire concatenated key for its existence
* If any column depends only on one part of the concatenated key, then the table fails second normal form test

|  |  |  |
| --- | --- | --- |
| **Student\_id** | **Student\_subject** | **Student\_age** |
| 101 | Maths | 20 |
| 101 | Physics | 20 |
| 102 | English | 21 |
| 103 | Computer | 22 |
| 103 | Chemistry | 22 |

The table is in 1NF but not in 2NF as non-prime attribute student\_age is dependent on student\_id which is a subset of candidate key. Hence it is broken down into two tables

Student\_details table:

|  |  |
| --- | --- |
| **Student\_id** | **Student\_age** |
| 101 | 20 |
| 102 | 21 |
| 103 | 22 |

Student\_subject table:

|  |  |
| --- | --- |
| **Student\_id** | **Student\_subject** |
| 101 | Maths |
| 101 | Physics |
| 102 | English |
| 103 | Computer |
| 103 | Chemistry |

**3NF form:**

* Every non‐prime attribute of table must be dependent on primary key
* Meet all the requirements of the second normal form AND no non‐key fields depend on a field(s) that is not the primary key
* Remove columns that are not dependent upon the primary key

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Emp\_id** | **Last\_name** | **First\_name** | **Dept\_num** | **Dept\_name** |
| 1001 | Pinto | Lee | 2 | Sales |
| 1005 | Wayne | Bruce | 2 | Sales |
| 1029 | Dsouza | John | 1 | Marketing |

Information about department name is repeated and is liable to become inconsistent. The name of department is determined by more than one field – If empID = 1001 then department name is Sales – If dept\_num is 2 is known, then it also yields Sales – There are two different fields determining the value of the department name

**To convert into 3NF:**

* Remove the field dept\_name from original Employee table and putting it in a new table along with the field on which it depends (dept\_num)
* The field dept\_num will be the primary key of the new table and will also remain in the Employee table as a foreign key

|  |  |  |  |
| --- | --- | --- | --- |
| **Emp\_id** | **Last\_name** | **First\_name** | **Dept\_num** |
| 1001 | Pinto | Lee | 2 |
| 1005 | Wayne | Bruce | 2 |
| 1029 | Dsouza | John | 1 |

|  |  |
| --- | --- |
| **Dept\_num** | **Dept\_name** |
| 1 | Sales |
| 2 | Marketing |
| 3 | Research |

Normalization is part of successful database design. Without normalization, database systems can be inaccurate, slow, or inefficient. Normalizing database is important as if not normalized they may not produce the data you expect.

The primary goals of normalizing data are:

* Arranging data into logical groupings such that each group describes a small part of the whole - for eg consider the person table, it only represents information related to a person and no unrelated data.
* Minimizing the amount of duplicate data stored in a database – for eg: an employee may have multiple entries related to subjects taken which can then be stored into multiple tables.
* Building a database in which you can access and manipulate the data quickly and efficiently without compromising the integrity of the data in storage – for eg if you update the patient table any corresponding table dependent on patient table will get updated.

When not to use normalization:

* Performing joins on tables is very expensive. Less number of joins which means select queries work faster. Normalizing the data would increase the number of tables.
* A single table with all the required data allows efficient index usage. If the columns are indexed properly, then results can be filtered and sorted by utilizing the same index. While in the case of a normalized table, data would be spread out in different tables so efficient usage of indexes won’t be possible.

References: Lecture slides on normalization

<http://www.ovaistariq.net/199/databases-normalization-or-denormalization-which-is-the-better-technique/>

[http://sqlmag.com/database-performance-tuning/sql-design-why-you-need-database--\*/normalization](http://sqlmag.com/database-performance-tuning/sql-design-why-you-need-database--*/normalization)

**9) Describe the process and implementation of a database performance tuning?**

Database tuning could be a difficult task especially when working with large-scale data where even the most minor change can have a positive or negative impact on performance.Even the performance of well-designed systems can degrade with use. Ongoing tuning is, therefore, an important part of proper system maintenance. For this, the database needs to be tuned properly using various tools.

Steps involved in the tuning process.

**Tune the Business Rules** - For optimal performance it is essential to adapt business rules. Configuration issues are considered at this level, such as whether to use a multi-threaded server system-wide. The planners ensure that the performance requirements of the system correspond directly to concrete business needs.**Tune the Data Design** –Determine what data is needed by your applications, what relations are important, and what their attributes are. Also structure the information to best meet performance goals. **Tune the Application Design** – Business executives and application designers should translate business goals into an effective system design. **Tune the Logical Structure of the Database** –Fine-tune the index design so that data to avoid over- nor under-indexed. In the data design phase identify the primary and foreign key indexes. In the logical structure design create additional indexes. **Tune Database Operations** –Use features such as Array processing, The Oracle optimizer, The row-level lock manager and PL/SQL based on the needs of your application:**Tune the Access Paths** – Use clusters, hash clusters, B\*-tree indexes, bitmap indexes, and optimizer hints for efficient data access. Also consider analyzing tables and using histograms.**Tune Memory Allocation** – Allocating memory resources to Oracle memory structures will have a positive effect on performance.

References:

<http://docs.oracle.com/cd/A87860_01/doc/server.817/a76992/ch2_meth.htm>

[https://docs.oracle.com/cd/E11882\_01/server.112/e41573/perf\_overview.htm#PFGRF02501](https://docs.oracle.com/cd/E11882_01/server.112/e41573/perf_overview.htm%23PFGRF02501)

|  |
| --- |
|  |