Data and Database Administration

- **Data Administration**: A high-level function that is responsible for the overall management of data resources in an organization, including maintaining corporate-wide definitions and standards.

- **Database Administration**: A technical function that is responsible for physical database design and for dealing with technical issues such as security enforcement, database performance, and backup and recovery.

Maintaining Security and Integrity of a Database:

- Security – protecting against unauthorized users
- Integrity – protecting against authorized users

<table>
<thead>
<tr>
<th>Data Administration</th>
<th>Database Administration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data policies, procedures, standards Planning</td>
<td>Selection of hardware and software</td>
</tr>
<tr>
<td>Data conflict (ownership) resolution</td>
<td>Installing/upgrading DBMS</td>
</tr>
<tr>
<td>Internal marketing of DA concepts</td>
<td>Tuning database performance</td>
</tr>
<tr>
<td>Managing the data repository</td>
<td>Improving query processing performance</td>
</tr>
<tr>
<td></td>
<td>Managing data security, privacy, and integrity</td>
</tr>
<tr>
<td></td>
<td>Data backup and recovery</td>
</tr>
</tbody>
</table>

**Protecting Data Integrity**

**SQL Mechanisms:**

- **Domain** – a new datatype – enforces field value integrity (not supported by Oracle).

  ```sql
  CREATE DOMAIN Gender AS CHAR(1) CHECK (VALUE IN ('M', 'F'));
  CREATE DOMAIN TemperatureF AS Number(3) CHECK (VALUE BETWEEN -50 AND 105 );
  ```

  Advantage over the CHECK – defined in one place.

- **Assertion** – conditions that DBMS checks and alerts the user when they are violated (not supported by Oracle).
  - Checked when mentioned relations change
CREATE ASSERTION ShipmentAssertion
CHECK NOT EXISTS
(SELECT shipmentNbr
 FROM Warehouse, Shipment
 WHERE ship_warehouse_nbr = warehouse_nbr AND
   warehouse_state != ship_destination_state)

- **Trigger** – automatically executed procedure, guards against/handles certain conditions occurring at the time of DB events (Update. Delete. Insert).

  Can be used to cause special handling procedures, recording certain values, e.g.
  - a large sale triggers a discount in price

  Can be used for integrity/security maintenance:
  1. Detecting fraud
     - logging the details (username, date/time) of a transaction related to sensitive data
     - prohibit inappropriate/erroneous actions
       *e.g. price drop exceeds the one-time discount range* 

SQL Details: there are two special variables new and old, representing the new and old record. Old makes no sense in an insert, and new makes no sense in a delete.

Notice: in WHEN new and old are used without a colon, but in actions, a preceding colon is needed.

```
CREATE TABLE U (
  who varchar(20),
  updateDate date,
  oldval number(3),
  newval number(3)
Constraint Primary Key (who, updateDate, oldval, newval) );

CREATE TRIGGER UpdateTrig
  AFTER UPDATE OF Number_Ordered ON order_line
  FOR EACH ROW
  WHEN(new.Number_Ordered > old.Number_Ordered)
BEGIN
  INSERT INTO U
  VALUES(User, Sysdate, :old.Number_Ordered,
       :new.Number_Ordered);
END;
```

run
```
update order_line
set number_ordered = number_ordered + 3
where order_number = 12498;
```
Another example: prevents unauthorized deletion.

```sql
CREATE TRIGGER DeleteTrig
BEFORE DELETE ON order_line
DECLARE
    weekend_error EXCEPTION;
    unauthorized_user EXCEPTION;
BEGIN
    IF TO_CHAR(SysDate, 'DY') = 'SAT' OR
       TO_CHAR(SysDate, 'DY') = 'SUN'
    RAISE weekend_error;
    END IF;

    IF User <> 'tbabaian' THEN
    RAISE unauthorized_user
    END IF;

EXCEPTION
    WHEN weekend_error THEN
        RAISE_APPLICATION_ERROR (-20001, 'Deletion on weekend not allowed');
    WHEN unauthorized_user THEN
        RAISE_APPLICATION_ERROR (-20002, 'Deletion by unauthorized user');
END;
```

Limitations:
In Oracle: the action cannot change the relation that triggers the action or a relation connected to the triggering relation by a foreign-key constraint

Authorization:

Restrict access to data with Views
Restrict actions (insert/update/delete/create/index/…) with Grant/Revoke privileges:

```
GRANT action ON table TO username1, ..., username1
```

Password protection
can be broken

Ensure network protection with encryption.
Concurrency Control

In a multi-user environment, simultaneous access to data can result in interference and data loss.

- **Solution – Concurrency Control**
- The process of managing simultaneous operations against a database so that data integrity is maintained and the operations do not interfere with each other in a multi-user environment.

PROBLEMS:

1. **Lost Update:**
   Transaction A’s update is lost, because transaction B overwrites it, without seeing A’s update.

<table>
<thead>
<tr>
<th>Transaction A:</th>
<th>Transaction B</th>
</tr>
</thead>
<tbody>
<tr>
<td>withdraw $10 from some account</td>
<td>withdraw $20 from the same account</td>
</tr>
</tbody>
</table>

   1. Retrieve R (balance is $100)
   2. Retrieve R (balance is $100)
   3. Update R (balance is $90)
   4. Update R (balance is $80)

2. **Uncommitted Dependency**

   Transaction A sees/uses the value that is later rolled back

<table>
<thead>
<tr>
<th>Transaction A:</th>
<th>Transaction B</th>
</tr>
</thead>
<tbody>
<tr>
<td>check balance</td>
<td>failed withdrawal from the same account</td>
</tr>
</tbody>
</table>

   Originally, balance = $100
   1. Update R (balance = $80)
   2. Retrieve R (balance = $80)
   3. Rollback (balance = $100)

Solutions to the problems of concurrent access: **Locking** and **Versioning**

1. **Locking**
   a. **Exclusive lock (X)** - No access permitted. Used when preparing to update
   b. **Shared lock (S)** – Weaker than the Exclusive. Read but no update permitted, i.e. when a shared lock (S) is placed on a record, another transaction can also place an S lock, but cannot place an X lock. Used when just reading to prevent another user from placing an exclusive lock (X) on the record.

   **Practice:** Check what happens with the two scenarios above if locking is applied.
Deadlock

A situation that results when two or more transactions have locked common resources, and each waits for the other to unlock their resources.

Managing deadlocks

- **Deadlock prevention:**
  - Lock all records required at the beginning of a transaction
  - Two-phase locking protocol
    - Growing phase
    - Shrinking phase
  - May be difficult to determine all needed resources in advance

- **Deadlock Resolution:**
  - Allow deadlocks to occur
  - Monitor the locks to detect and break deadlocks (using cycle-detection in *Wait-For-Graphs*, for instance)

  Breaking the deadlock involves:
  - Selecting one transaction as a *victim*.
  - Rolling the *victim* back – sending a message to the application that issued the transaction, so that it can be restarted.

2. **Versioning** – an optimistic approach to concurrency control, assumes that simultaneous updates will be infrequent

- Instead of locking each transaction can attempt an update as it wishes.
- Transactions are timed, when the DBMS receives the results of each it will reject an update when there is a conflict:

Example:

<table>
<thead>
<tr>
<th>Transaction A:</th>
<th>Transaction B</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>withdraw $10 from some account</em></td>
<td><em>withdraw $20 from the same account</em></td>
</tr>
</tbody>
</table>

1. Retrieve R (balance is $100)  
2. Retrieve R(balance is $100)  
3. Update R (balance is $90)  
4. Update R(balance is $80)  
5. Commit                      
   Rollback and Restart

Here Transaction B will initially be allowed to go through, but at time 4, since it used a value of balance at the time that another transaction was making an update.
Backup and Recovery:

- Mechanism for restoring a database quickly and accurately after loss or damage/

Automatic dump facility that produces backup copy of the entire database
Periodic backup (e.g. nightly, weekly)
  Cold backup – database is shut down during backup
  Hot backup – selected portion is shut down and backed up at a given time
Backups are stored in secure, off-site location

Audit trail of transactions and database updates
  **Transaction log** – record of essential data for each transaction processed against the database
  **Database change log** – images of updated data
    - Before-image – copy before modification
    - After-image – copy after modification

Periodic synchronization of the transaction logs and the database enables recovery manager to resume processing from short period, instead of repeating the entire day’s worth of transactions.

**Checkpoint** –
a timed COMMIT.
Recovery and Restart procedures. Definitions

- Switch - to a mirrored database
- Restore/Rerun - Reprocess transactions against the backup
- Backward Recovery (Rollback) - Apply *before-images* to the current database
- Forward Recovery (Roll Forward) - Apply *after-images* to the database restored to its original state (restored)

Database Failures and Responses:

1. *Aborted transactions*
   a. Preferred recovery: rollback
   b. Alternative: Rollforward to state just prior to abort

2. *Incorrect data*
   a. Preferred recovery: rollback
   b. Alternative 1: re-run transactions not including inaccurate data updates
   c. Alternative 2: compensating transactions

3. *System failure (database intact)*
   a. Preferred recovery: switch to duplicate database
   b. Alternative 1: rollback
   c. Alternative 2: restart from checkpoint

4. *Database destruction*
   a. Preferred recovery: switch to duplicate database
   b. Alternative 1: rollforward
   c. Alternative 2: reprocess transactions

Database Performance Tuning

- tasks performed by a Database Administrator

**DBMS Installation**
  d. Setting installation parameters

**Monitor and configure Memory Usage**

**Monitor and adjust Input/Output Contention**
  e. Distribution of heavily accessed files

**Monitor CPU load**

**Application tuning:** modification of SQL code in applications