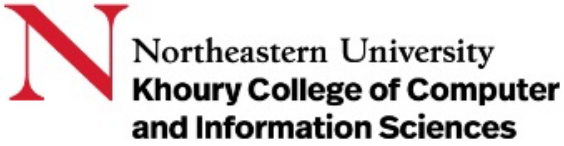
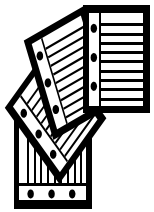
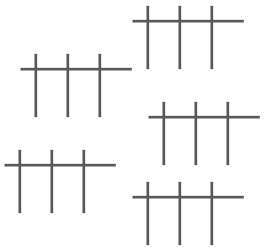


Architecture:
Triggers & Stored Procedures

Copyright © 2011-2020 by Khoury College of Computer Sciences.
No Duplication or Distribution Without the Express Permission of Khoury.

Systems Analysis Lifecycle





Enterprise Analysis

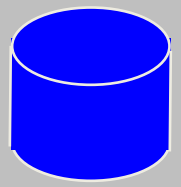
Requirements Analysis

Conceptual Data Modeling

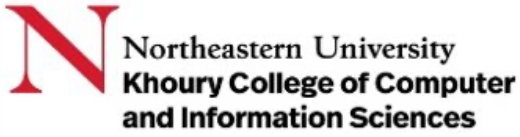
Logical Data Modeling

Use Case Analysis

Prototyping and RAD



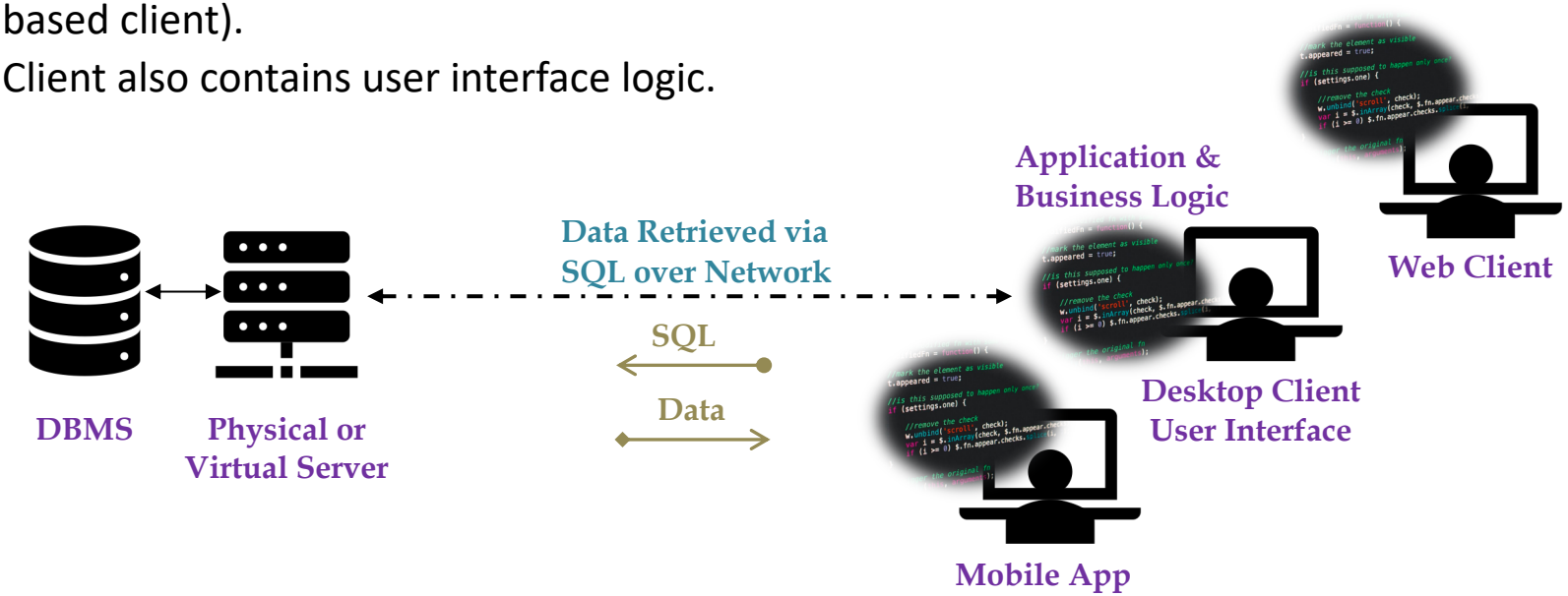
Physical Database Design



Multi-Tier Data Architectures

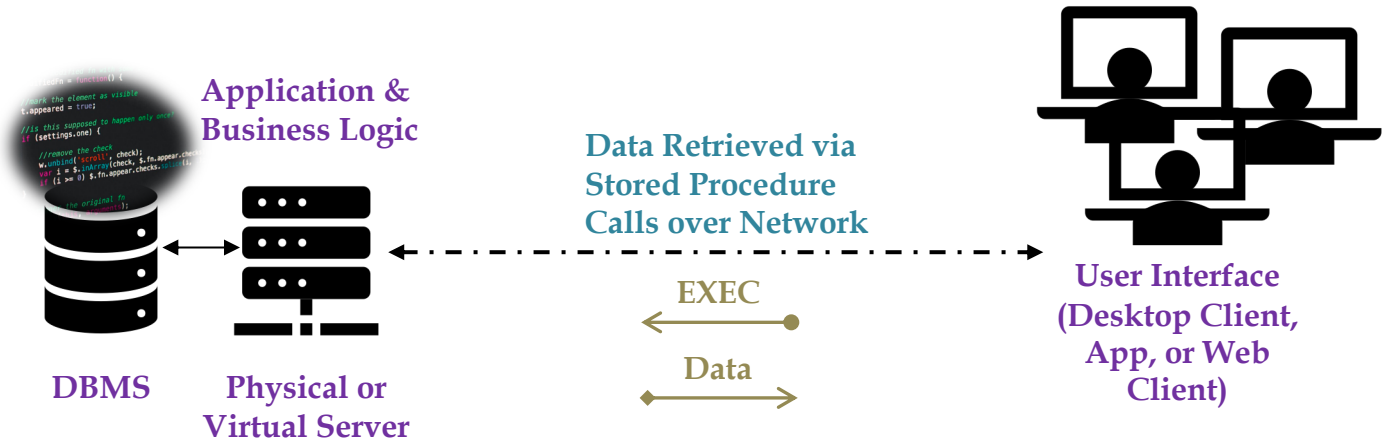
Two-Tier Architecture with Business Logic on the Client

- Data and business logic is distributed two tiers. The data is within a DBMS housed on a server while the user interacts through a client interface.
- This client/server architecture is often called a “fat client” architecture.
- Business logic, including constraint and integrity checking and calculations, are performed on the client (either a desktop client application, a mobile app, or a browser-based client).
- Client also contains user interface logic.



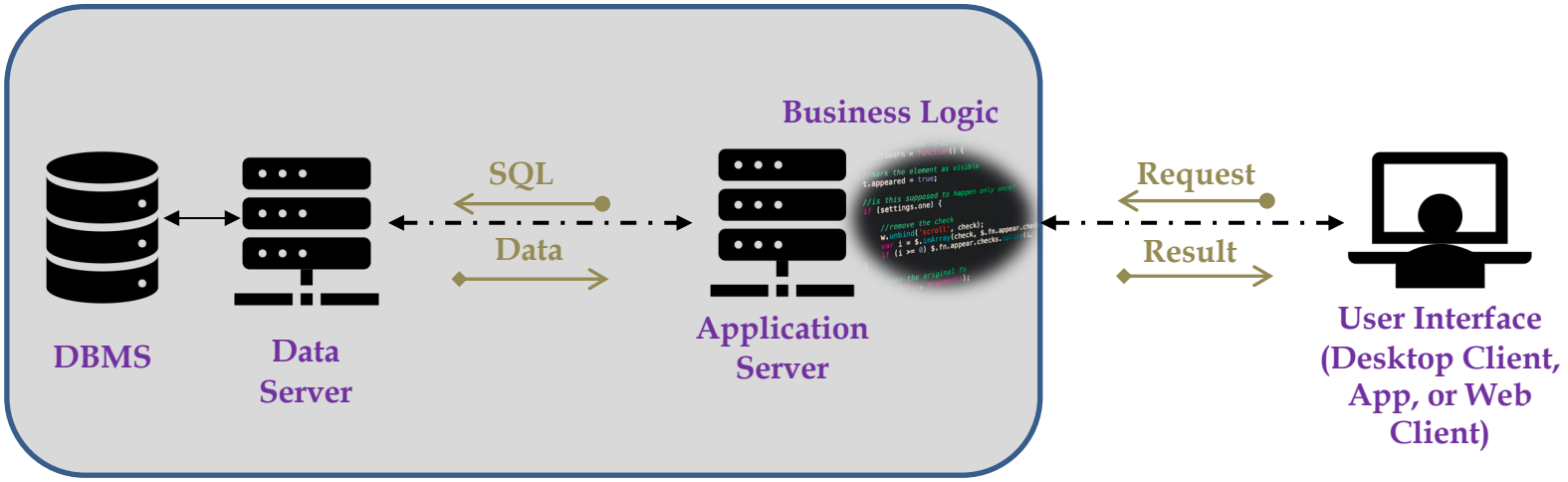
Two-Tier Architecture with Business Logic on the Server

- Data and business logic is distributed two tiers. The data is within a DBMS housed on a server while the user interacts through a client interface.
- This client/server architecture is often called a “thin client” architecture.
- Business logic, including constraint and integrity checking and calculations, are performed on the database within the server using triggers and stored procedures.
- Client hosts user interface logic only.



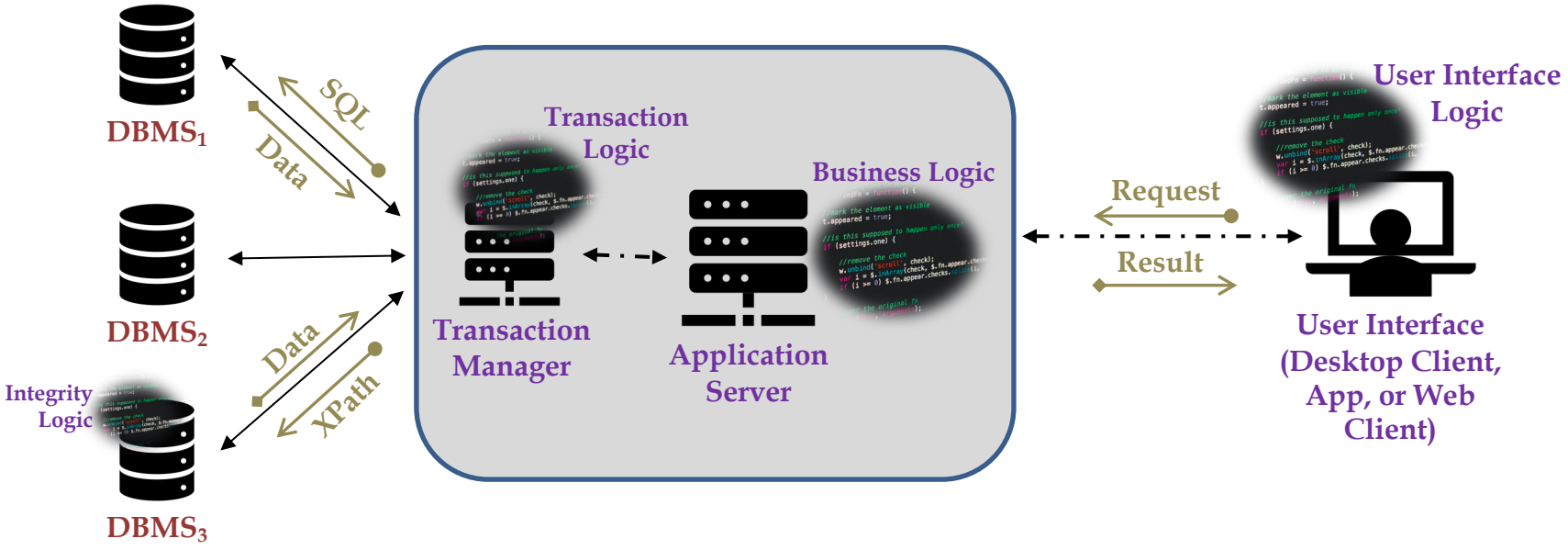
Three-Tier Architecture with Application Server

- Data is on data server and all business logic resides on an application server.
- Communication between the “thin client” is done through remote procedure invocation technology such as WebRPC, SOAP, or custom communication protocols.



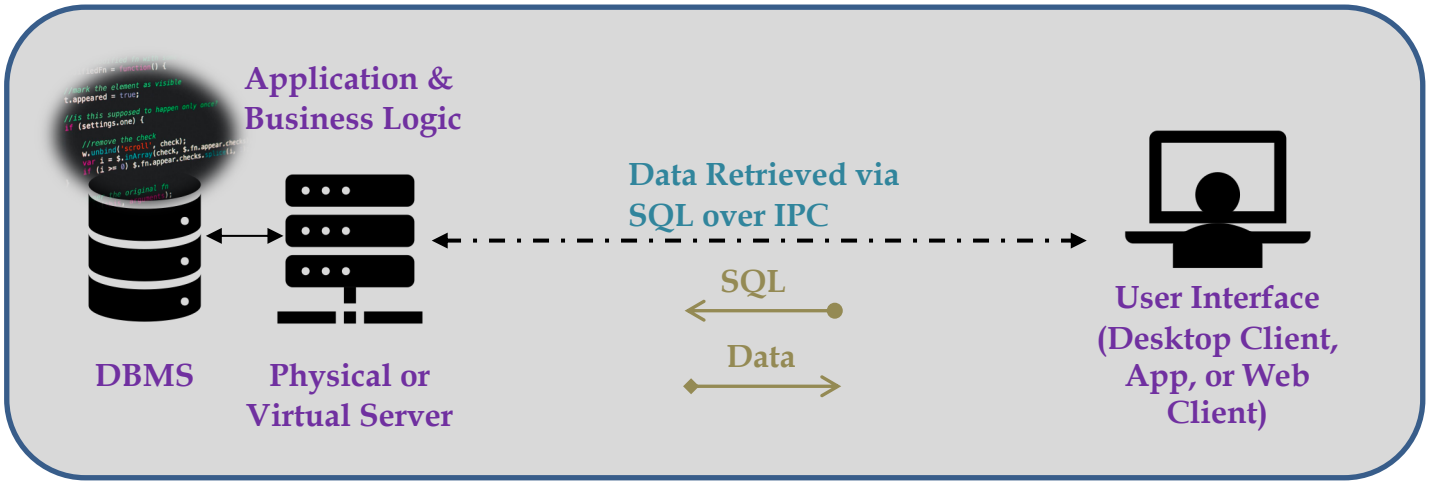
n-Tier Distributed Architecture

- Data and business logic is distributed across tiers using combinations of triggers, stored procedures, and business logic on an application server.
- This is the preferred architectural model if the data is distributed across several (possibly disparate) databases.
- A transaction manager often handles distributed transaction logic, although this logic is often co-located on the application server (logical tiers).



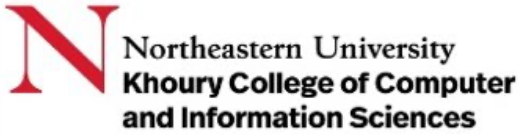
Physical vs Logical Tiers

- The tiers do not have to be physical tiers, i.e., the server does not have to be a separate computer or virtual machine.
- A single computer or virtual machine can host all the tiers leading to a logical two-tier or multi-tier architecture.



Logical Two-Tier Client/Server Architecture with a Thin Client

Summary, Review, & Questions…



Triggers (SQLite)

Triggers

A trigger is a database program object fired automatically when data in a table is changed in some way either through a delete, and update, or an insert.

Triggers are associated with a specific table and a specific action on the associated table.

Triggers can be defined to “fire” before or after some action on the table.

Triggers cannot be associated with a selection (retrieval) operation.

Common uses of triggers include:

- checking integrity and other user-defined constraints
- auditing of updates to data, *e.g.*, logging changes to sensitive data whenever they occur
- enforcing complex business rules centrally in the database
- preventing invalid actions on the data
- recalculating the value of a derived attribute

SQLite **CREATE TRIGGER** Statement

- To create a new trigger in *SQLite*, you use the `CREATE TRIGGER` statement:

```
CREATE TRIGGER [IF NOT EXISTS] trigger_name
  [BEFORE|AFTER|INSTEAD OF] [INSERT|UPDATE|DELETE] ON table
  [WHEN condition]
  BEGIN
    statements;
  END;
```

SQLite Trigger Definition Syntax & Limitations

1. Specify the name of the trigger after the `CREATE TRIGGER` keywords.
2. Determine when the trigger is fired: `BEFORE`, `AFTER`, or `INSTEAD OF` the action on the table. Note that `INSTEAD OF` triggers on apply to a view and not a table.
3. Specify the table to which the trigger belongs.
4. Place the trigger logic in between the `BEGIN/END` block, using any valid SQL statements.

➢ Specifying a condition in the `WHEN` clause, invokes the trigger only invoked when the condition is true. Omitting the `WHEN` clause, executes the trigger for all affected rows.

- *SQLite* is limited in its support for programming the logic of a trigger.
- Unlike other DBMS implementations it does not support loops, variables, or conditional logic.
- As of version 3.9.2, *SQLite* only supports `FOR EACH ROW` triggers; it does not yet support `FOR EACH STATEMENT` triggers.

A trigger is deleted with the `DROP TRIGGER` statement.

Note that if a table is deleted (`DROP TABLE`), then all associated triggers are deleted, unless the trigger references other tables.

TRIGGER DELETION

Accessing Old and New Data

Access to the data of the row being inserted, deleted, or updated is through the `OLD` and `NEW` references in the form:

- `OLD.column_name`
- `NEW.column_name`

The `OLD` and `NEW` references are available depending on the event that caused the trigger to be fired:

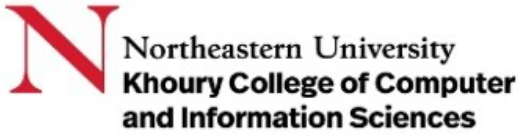
Action	Reference Available
`INSERT`	`NEW`
`DELETE`	`OLD`
`UPDATE`	`NEW` and `OLD`

```
CREATE TABLE leads (
        id integer PRIMARY KEY,
        first_name text NOT NULL,
        last_name text NOT NULL,
        phone text NOT NULL,
        email text NOT NULL,
        source text NOT NULL
);
```

```
CREATE TRIGGER validate
  BEFORE INSERT ON leads
  BEGIN
    SELECT
      CASE
        WHEN NEW.email NOT LIKE '%_@__%.__%'
        THEN RAISE (ABORT,'Invalid email')
      END;
  END;
```

Adapted from https://www.sqlitetutorial.net/sqlite-trigger

Summary, Review, & Questions…



Stored Procedures

Summary, Review, & Questions…

Key Topics

- Mapping to Tables
- Multiplicity Mapping Rules
- Link Tables
- Mapping Generalizations
- Managing Normal Forms