Database Design Process

Real World

Requirements Analysis

Functional Requirements

Functional Analysis

Access Specifications

E-R Modeling

Choice of a DBMS

Conceptual Design

Logical Design

Logical Schema

Physical Design

Requirements Collection & Analysis

Examples of activities:

- Identification of user groups and application areas
- Analysis of the operating environment and processing requirements
- Interviews

Caveat:

- Users change their minds
- Anticipating users’ future desires is difficult
- On the one hand: Adaptive system design is good.
- On the other hand: Good performance requires freezing important system parameters.
Conceptual Database Design

- Conceptual Schema Design:
  - Database structure, semantics, interrelationships, and constraints.
  - A stable description of the database (anticipating users’ desires).
  - High-level data model may be useful:
    - Expressiveness
    - Simplicity
    - Minimality
    - Diagrammatic

Design strategies

- **Top-down**: start from abstraction and use successive refinements.
  - This is the one we focus on
- **Bottom-up**: start from concrete designs to find abstractions.
  - Databases exist; the focus is on integration
- **Iterative**: mixed top-down and bottom-up as appropriate
Logical Database Design

- Data model mapping
- Convert the conceptual and external models into the DBMS’s high-level data model.
  - The result of this phase is a set of DDL statements in the language of the chosen DBMS.

Physical Database Design

- Storage structures and access paths
- General user requirements – examples:
  - Response-time: 95% of transactions must answer within 2 seconds
  - Space utilization: disk should not be more than half empty
  - Throughput: At peak times, must handle 1500 transactions per second
- Separate read-only queries from update transactions
  - Expected frequency of queries and transactions.
  - User requirements on response-time and throughput
  - Optimization techniques:
    ➤ Denormalization, duplication
    ➤ Indexed files for scan and hashing for random access
Conceptual Design

Entity-Relationship Modeling

- **Top-Down Design**
  - Determine the entities, attributes, relationships
  - Model them properly
  - Map the resulting E-R model into a data model
- **Conceptual**
  - No physical details
  - Easier to detect conceptual design errors
- **One of the logical database design aids**
  - Significant amount of research within the database community
  - Easy mappings to other data models possible
Entity-Relationship Modeling

- **Entity**
  - An object that *exists* in the real world, that has certain *properties* and that is *distinguishable* from other objects
  - Example
    - Employee
    - Project

- **Relationship**
  - Associations between two or more entities
  - Example
    - Manage: Employees manage projects
    - Work: Employees work in projects

- **Attribute**
  - The properties of entities and relationships
  - Example
    - Employee: Employee No, Name, Title, Salary
    - Work: Responsibility, Assignment duration

Entity Types and Instances

- **Entity type** is an abstraction that defines the properties (attributes) of a similar set of entities
  - Example:
    - Employee: Name, Title, Salary
    - Project: Name, Budget, Location

- **Entity instances** are instantiations of types
  - Example:
    - Employee: Joe, Jim, ...
    - Project: Compiler design, Accounting, ...

- An entity instance can have multiple entity types
  - Example:
    - If we also want to have an EMPLOYEE entity type, then every engineer is also an employee

- **Entity class** (or *entity set*) is a set of entity instances that are of the same type
  - Similar arguments can be made for relationships
Types and Instances

<table>
<thead>
<tr>
<th>Employee</th>
<th>Works On</th>
<th>Project</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1 J. Doe</td>
<td></td>
<td>P1 Instrumentation</td>
</tr>
<tr>
<td>E2 M. Smith</td>
<td></td>
<td>P2 Database Develop.</td>
</tr>
<tr>
<td>E3 A. Lee</td>
<td></td>
<td>P3 CAD/CAM</td>
</tr>
<tr>
<td>E4 J. Miller</td>
<td></td>
<td>P4 Maintenance</td>
</tr>
<tr>
<td>E5 B. Casey</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E6 L. Chu</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E7 R. Davis</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E8 J. Jones</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Attributes

- Describe properties of entities and relationships
- An instance of an attribute is a value, drawn from given domain, which presents the set of possible values of the attribute.
- Types:
  - Single vs multivalued
    - Single: Social insurance number
    - Multi: Lecturers of a course
  - Simple vs composite
    - Composite: Address consisting of Apt#, Street, City, Zip
  - Stored vs derived attribute
    - Stored: Individual mark of a student
    - Derived: Average mark in a class
  - Key attribute - identifier
Identifiers

- **Entity identifier**
  - One or more of the attributes that can uniquely identify each instance of a given entity type
  - Example
    - Employee: Employee No
    - Project: Project No

- **Relationship identifier**
  - A means of identifying each relationship instance.
  - Usually a composite identifier consisting of the identifiers of the two or more entity types that it relates
  - Example
    - Works(Employee No, Project No)

Entities-Attributes-Relationships

- SUPPLIER
  - Supplier No
  - Supplier Name
  - Location
  - are supplied by

- WAREHOUSE
  - Wareh. No
  - Wareh. Name
  - Location
  - are supplied by

- PART
  - Part No
  - Part Name
  - QTY
  - WGT
  - are used by

- PROJECT
  - Project No
  - Project Name
  - Budget
  - work on

- EMPLOYEE
  - Emp. No
  - Emp. Name
  - Addr
  - Title
  - Salary
  - employ

- LOCATION
  - City

- DEPARTMENT
  - Dept. No
  - Dept. Name
  - Manager

are supplied by
supply

are supplied by
are used by

supply
use

work on

employ

manage
E-R Notation

Entity types and instances

Attributes

Relationships

E-R Diagrams

EMPLOYEE

PROJECT

WORKS ON

Employee No

Employee Name

Duration

Project No

Project Name

Title

Salary

Responsibility

Budget

NoEmp

Address

Apt. #

City

Street #

Location
Semantics of E-R Models

- There is a need to capture the semantics of entities and relationships
- This is done by means of constraints
  - Primary Key: One of the identifiers of each entity and relationship
  - Cardinality constraints: types of relationships
  - Existence (participation) constraint
  - Referential integrity

Types of Relationships

- Fundamental ones
  - One-to-one
  - Many-to-one (one-to-many)
  - Many-to-many
- Recursive relationships

There can be multiple relationships between two entity types
Each instance of one entity class E1 can be associated with **at most one** one instance of another entity class E2 and vice versa.

**Example:**
- Each employee can work in **at most one** project and each project employs **at most one** employee.
Many-to-One Relationship

- Each instance of one entity class E1 can be associated with zero or more instances of another entity class E2, but each instance of E2 can be associated with at most 1 instance of E1.
- Example:
  - Each employee can work in at most one project; each project can employ many engineers.
Many-to-Many Relationship

- Each instance of one entity class can be associated with many instances of another entity class, and vice versa.
- Example:
  - Each employee can work in many projects; each project can employ many employees

```
<table>
<thead>
<tr>
<th>Employee No</th>
<th>Employee Name</th>
<th>Duration</th>
<th>Responsibility</th>
<th>Salary</th>
<th>Title</th>
<th>Project No</th>
<th>Project Name</th>
<th>Budget</th>
</tr>
</thead>
</table>
```

**WORKS_ON**

- Relationship Instances
- EMPLOYEE Set
- PROJECT Set

3-23

3-24
Recursive Relationships

- An entity instance of type $T_1$ is in a relationship with another entity instance of type $T_1$.
- It assumes multiple roles.

Multiple Relationships
Higher-Order Relationships

A relationship can link more than one type of entity.

Constraints

- Referential integrity
  - When there is a 1:1 or M:1 relationship R between entity types E1 and E2, if one and exactly one instance of E2 has to exist for a given instance of E1, a referential integrity constraint exists.

- Participation constraint
  - Determines whether instances of a given entity can exist without participating in a relationship.

- Cardinality constraint
  - Relationship types (1:1, M:1, M:N) and their refinement where the exact number is specified.
Participation Constraints

Whether or not the existence of an entity depends on its being related to another entity via the relationship type

- Total: If entity \( E_i \) is in total participation with relation \( R \), then every entity instance of \( E_i \) has to participate via relation \( R \) to an entity instance of another entity type \( E_j \)
- Partial: Only some entity instances participate

Referential Integrity

Assume that for a given project, there has to be one and only one employee managing it
Strong and Weak Entity Sets

**Strong entities**: The instances of the entity class can exist on their own, without participating in any relationship.
- Also called *non-obligatory membership*.

**Weak entities**: Each instance of the entity class has to participate in a relationship in order to exist. Keys are imported from dependent entity.
- Also called *obligatory membership*.
- Special type of total participation

---

Connection Traps

- Be careful in defining and interpreting relationships.
- For example, consider the following diagram.
  - Can we find, for any given employee, which department he is in?
  - Conversely, can we find, for a given department, which employees are in that department?
Connection Traps

One solution is to change the relationship definition.

Connection Traps

What will happen if some employees are connected with divisions (e.g., as consultants to division heads), but are not included in any department?
Simplifications

- Sometimes it is necessary to simplify some of the relationships
  - Some older data models cannot handle them
    - Even object models sometimes require relationships to be binary
  - Some E-R based database design tools permit binary relationships only

- Types of simplifications
  - Many-to-many $\Rightarrow$ Two one-to-many
  - Higher order relationships $\Rightarrow$ binary relationships

- Simplification is done by creating new relationships
- Connection traps cause significant difficulties

Many-to-Many Simplification

- Can not do by simple creation of two 1:N relationships between the two entity classes.
  - N:M relationship indicates that there is no dependence between the instances of the two entity classes.
  - 1:N relationship forces a dependency.
- Consider N:M relationship between EMPLOYEE and PROJECT
Many-to-Many Simplification

- Treat the relationship as an entity class. Define suitable relationships among three entities.
- This simplification is not necessary for mapping into the relational model, but is important for mapping into other models.

![Diagram of Many-to-Many Relationships]

Higher-Order Relationships

A relationship can link more than one type of entity.

![Diagram of Higher-Order Relationships]
Higher-Level Relationships

Create an intermediate weak entity type

Specialization

- An entity type \(E_1\) is a specialization of another entity type \(E_2\) if \(E_1\) has the same properties of \(E_2\) and perhaps even more.
- \(E_1\) IS-A \(E_2\)
Attribute Inheritance

Specialization
Subclass/Superclass

- This is related to instances of entities that are involved in a specialization/generalization relationship
- If E1 specializes E2, then each instance of E1 is also an instance of E2. Therefore
  \[ \text{Class}(E_1) \subseteq \text{Class}(E_2) \]
- Example
  \[ \text{Class}(\text{Manager}) \subseteq \text{Class}(\text{Employee}) \]
  \[ \text{Class}(\text{Employee}) \subseteq \text{Class}(\text{Engineer}) \cup \text{Class}(\text{Secretary}) \cup \text{Class}(\text{Salesperson}) \]

Specialization Constraints

- Disjoint
  - Entity instances in a subclass cannot exist in more than one subclass
  - E.g., an employee cannot be a secretary and an engineer at the same time
- Overlapping
  - Entity instances can be members of multiple subclasses
  - E.g., an object can both be manufactured and purchased
Specialization Constraint Combinations

- disjoint, total
- disjoint, partial
- overlapping, total
- overlapping, partial

Total & Partial Disjoint

Diagram showing the relationship between different types of employees and their associated attributes such as title, address, salary, and specialty.
Total Overlapping

Aggregation
Design Process - Where are we?

Conceptual Design

Conceptual Schema (ER Model)

Logical Design

Logical Schema (Relational Model)

Step 1: ER-to-Relational Mapping

Example

Diagram showing entities and relationships:

- EMPLOYEE
- PROJECT
- MANUFACTURED_PART
- PURCHASED_PART
- MANAGER
- WORKS_ON
- ACCOUNT
- RECORD
- SUPPLIER
- SUPPLY
- PART
- CONTAIN

Example data:

<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>PROJECT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee No</td>
<td>Project No</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Salary</td>
<td>Budget</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>PART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project No</td>
<td>Part No</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
<tr>
<td>Location</td>
<td>Location</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PART</th>
<th>ACCOUNT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part No</td>
<td>Account #</td>
</tr>
<tr>
<td>Name</td>
<td>Income</td>
</tr>
<tr>
<td>Color</td>
<td>Expenses</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACCOUNT</th>
<th>RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Account #</td>
<td>Balance</td>
</tr>
<tr>
<td>Income</td>
<td>Expenses</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SUPPLIER</th>
<th>SUPPLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier No</td>
<td>Amount</td>
</tr>
<tr>
<td>Name</td>
<td>Location</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MANUFACTURED_PART</th>
<th>PURCHASED_PART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch No</td>
<td>Drawing No</td>
</tr>
<tr>
<td>Price</td>
<td>Price</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MANAGER</th>
<th>WORKS_ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee No</td>
<td>Project No</td>
</tr>
<tr>
<td>Name</td>
<td>Name</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MANUFACTURED_PART</th>
<th>PURCHASED_PART</th>
</tr>
</thead>
<tbody>
<tr>
<td>Made-up of</td>
<td>Consists of</td>
</tr>
<tr>
<td>Qty</td>
<td>Qty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SALESPERSON</th>
<th>SECRETARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>Office</td>
</tr>
<tr>
<td>Region</td>
<td>Specialty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SALESPERSON</th>
<th>ENGINEER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>Office</td>
</tr>
<tr>
<td>Region</td>
<td>Specialty</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SALESPERSON</th>
<th>ENGINEER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>Office</td>
</tr>
<tr>
<td>Region</td>
<td>Specialty</td>
</tr>
</tbody>
</table>
Step 1 - Handling Entities

- For each regular entity type $E$ in the E-R schema, create a relation $R$.
  - Include as attributes of $R$ are only the simple attributes of $E$.
  - For composite attributes of $E$, just include their constituent simple attributes in $R$.
  - The key of $E$ becomes the primary key of $R$. If there are more than one key attributes of $E$, then choose one as the primary key of $R$.

Step 1 – Example

- Create the following relation schemes.
  - The keys are underlined.

  EMPLOYEE(ENO, ENAME, TITLE, SALARY, APT#, STREET, CITY)
  PROJECT(PJNO, PNAME, BUDGET)
  SUPPLIER(SNO, SNAME, CREDIT, LOCATION)
  PART(PNO, PNAME, WGT, COLOR)
Step 2 – Weak Entities

- For each weak entity type $W$ associated with the strong entity type $E$ in the E-R schema, create a relation $R$.
  - The attributes of $R$ are the *simple* attributes of $W$ (or the simplified versions of composite attributes).
  - Include among the attributes of $R$ all the key attributes of strong entity $E$. These serve as *foreign keys* of $R$.
  - The primary key of $R$ is the combination of the primary key of $E$ and the partial key of $W$.

Step 2 – Example

- Example:
  - Create relation ACCOUNT as follows
    \[
    \text{ACCOUNT(PJNO,ACNO,INCOME,EXPENSES)}
    \]
    
    foreign key
Step 3 – 1:1 Relationships

For each 1:1 relationship $R$ in E-R schema where the two related entities are $E_1$ and $E_2$. Let relations $S$ and $T$ correspond to $E_1$ and $E_2$ respectively.

- Choose one of the relations, preferably one whose participation in $R$ is total (say $S$). Include in $S$ the primary key of $T$ as a foreign key.
- Also, if there are attributes associated with the relationship $R$, include them in $S$.
- You may want to rename the attributes while you do this.

Step 3 – Example

For 1:1 relationship MANAGES between the EMPLOYEE and PROJECT entities.

- Choose PROJECT as S, because its participation in the MANAGES relationship is total (every project has a manager, but every employee does not need to manage a project). Then, include in PROJECT the primary key of EMPLOYEE.
  
  PROJECT(PJNO,PNAME,BUDGET,MGR)

FOR 1:1 relationship RECORDS between PROJECT and ACCOUNT entities:

- Choose ACCOUNT as S (note: ACCOUNT is a weak entity, so this is the only choice that makes sense)
- Include in ACCOUNT PJNO (which was done in step 2) and BALANCE
  
  ACCOUNT(PJNO,ACNO,INCOME,EXPENSES,BALANCE)
Step 4 – 1:N Relationships

- For each regular (non-weak) binary 1:N relationship type $R$ in the E-R schema identify the relation $S$ that corresponds to the entity type at the N-side of the relationship. Let the other relation on the 1-side be $T$.
  - Include in $S$ as foreign key the primary key of $T$.
  - If there are attributes associated with the relationship $R$, include them in $S$ as well.

Step 4 – Example

- We have only the WORKS ON relationship to consider. It is defined in between PROJECT and EMPLOYEE
  - N side of the relationship is EMPLOYEE; 1 side is PROJECT
  - Include in EMPLOYEE
    - Primary key (PJNO) of PROJECT
    - Attributes of the WORKS ON relationship (Duration & Responsibility

  EMPLOYEE(ENO, ENAME, TITLE, SALARY, APT#, STREET, CITY, PJNO, DURATION, RESP)
Step 4 – 1:N Relationships

- If this is a problem, then create a new relation $S$ corresponding to relationship $R$ and include in $S$ the primary keys of the two entities that $R$ links in addition to its own attributes. The primary key of $S$ is the combination of the primary keys of the two entities.

- In our case, we would have

```
WORKS(ENO, PJNO, DURATION, RESP)
```

Step 5 – M:N Relationships

- For each binary M:N relationship type $R$ connecting entities $E_1$ and $E_2$ in the E-R schema, create a relation $S$.
  - Include as foreign keys of $S$, the primary keys of the two relations that correspond to $E_1$ and $E_2$.
  - These attributes, together, form the primary key of $S$.
  - Also include in $S$ any attributes of the relationship $R$. 
Step 5 – Example

- We have one M:N relationship to consider: CONTAIN, which is a recursive relationship over the PART entity.
  - We create the following relation:
    \[ \text{CONTAIN(PNO1,PNO2,QTY)} \]

Step 6 – Multivalued Attributes

- For each multivalued attribute \( A \), create a new relation \( R \).
  - The attributes of \( R \) are \( A \) (if composite, then the simple components).
  - Also include in \( R \) the primary key \( K \) of the entity that contained \( A \).
  - The primary key of \( R \) then becomes \( K \) and \( A \) together.
Step 6 – Example

- In our example, we have to create one new relation for the multivalued attribute LOCATION in PROJECT.
  - This relation is created as follows:
    \[
    \text{LOC(PJNO,LOCATION)}
    \]

Step 7 – Higher Order Relationships

- For each higher order relationship type \( R \) connecting \( E_1, E_2, \ldots, E_n \) in the E-R schema, create a relation \( S \).
  - Include in \( S \) the primary keys of the relations corresponding to \( E_1, E_2, \ldots, E_n \).
  - Also include in \( S \) any attributes of \( R \).
  - The primary key of \( S \) is the combination of the primary keys of the relations corresponding to \( E_1, E_2, \ldots, E_n \).
Step 7 – Example

- The only high-order relation is SUPPLY between SUPPLIER, PROJECT and PART
  - Create relation SUPPLY
    
    \[
    \text{SUPPLY(SNO,PJNO,PNO,AMOUNT,DATE)}
    \]

Step 8 – Specialization

- For each specialization with \( m \) subclasses \( \{S_1, \ldots, S_m\} \) and generalized superclass \( C \), where the attributes of \( C \) are \( \{k, A_1, \ldots, A_n\} \) (\( k \) is the primary key), convert according to the following:
  1. General case:
    - Create a relation \( T \) for \( C \) with attributes \( \{k, A_1, \ldots, A_n\} \) and use \( k \) as the primary key.
    - Create one relation \( U_j \) for each \( S_j \). Include in \( U_j \) all the attributes of \( S_j \) and \( k \). Use \( k \) as the primary key of \( U_j \).
Step 8 – Specialization (cont’d)

③ No superclass relation:
- Create one relation \( U_i \) for each \( S_i \). Include in \( U_i \) all the attributes of \( S_i \) and \( \{ k, A_1, \ldots, A_n \} \). Use \( k \) as the primary key of \( U_i \).

④ For disjoint subclasses:
- Create a single relation \( U \) which contains all the attributes of all \( S_i \) and \( \{ k, A_1, \ldots, A_n \} \) and \( t \). Use \( k \) as the primary key of \( U \). The attribute \( t \) indicates the type attribute according to which specialization is done.

⑤ Note: May generate a large number of null values in the relation.

Step 8 – Specialization (cont’d)

④ For overlapping subclasses:
- Create a single relation \( U \) which contains all the attributes of all \( S_i \) and \( \{ k, A_1, \ldots, A_n \} \) and \( \{ t_1, \ldots, t_m \} \). Use \( k \) as the primary key of \( U \). The attributes \( t_i \) are boolean valued, indicating if a tuple belongs to subclass \( S_i \).
- Note: May generate a large number of null values in the relation.
Step 8 – Example

- Specialization of EMPLOYEE
  - Relation EMPLOYEE already exists; option 2 is not valid
  - Specialization is disjoint; option 4 is not valid
  - Options 1 or 3 are possible:
    - Option 1:
      
      ENGINEER(ENO, PROJECT, OFFICE)
      SECRETARY(ENO, OFFICE, SPECIALTY)
      SALESPERSON(ENO, CAR, REGION)
    - Option 3:
      
      EMPLOYEE(ENO, ENAME, TITLE, SALARY, APT#, STREET, CITY, PJNO, DURATION, RESP, TYPE, PROJECT, OFFICE, SPECIALTY, CAR, REGION)

Step 8 – Example (cont’d)

- Specialization of PART
  - Relation PART already exists; option 2 is not valid
  - Specialization is overlapping; option 3 is not valid
  - Options 1 or 4 are possible:
    - Option 1:
      
      MANUFACTURED_PART(PNO, BATCH#, DRAWING#)
      PURCHASED_PART(PNO, PRICE)
    - Option 4:
      
      PART(PNO, PNAME, WGT, COLOR, MAN, PURC, BATCH#, DRAWING#, PRICE)
Step 9 – Aggregation

- General case:
  - Treat the aggregation relationship as a normal relationship and map to a relation
- In our case we have no aggregation

Final Set of Relations

EMPLOYEE(ENO, ENAME, TITLE, SALARY, APT#, STREET, CITY, PJNO, DURATION, RESP)
PROJECT(PJNO, PNAME, BUDGET, MGR)
SUPPLIER(SNO, SNAME, CREDIT, LOCATION)
PART(PNO, PNAME, WGT, COLOR, MAN, PURC, BATCH#, DRAWING#, PRICE)
ENGINEER(ENO, PROJECT, OFFICE)
SECRETARY(ENO, OFFICE, SPECIALTY)
SALESPERSON(ENO, CAR, REGION)
SUPPLY(SNO, PJNO, PNO, AMOUNT, DATE)
LOC(PJNO, LOCATION)
CONTAIN(PNO, PNO, QTY)
ACCOUNT(PJNO, ACNO, INCOME, EXPENSES, BALANCE)