Databases

Lecture 8 - Views and Transactions

Emily Lucia Antosch

HAW Hamburg

14.03.2025

Contents

1. Introduction

1.1 Where are we right now?

- Last time, we looked at the basics of subqueries and views
- Today, we'll be discussing
  - how we can expand our knowledge of views,
  - how we can use transactions to increase the safety of our data manipulation statements
  - how transactions are executed.

1.1 Where are we right now?

1. Introduction
2. Basics
3. SQL
4. Entity-Relationship-Model
5. Relationships
6. Constraints
8. **Subqueries & Views**
9. **Transactions**
10. Database Applications
11. Integrity, Trigger & Security

1.2 What is the goal of this chapter?

- At the end of this lesson, you should be able to
  - create views in PostgresQL and use them effectively and
  - use transactions to make safe changes, that can be undone if necessary.

2. Repetition

2.1 Relational Algebra

Basics

- SQL is the What, while the Relational Algebra is the How!
- In mathematics an algebra is a values range combined with defined operations
- **Relational Algebra**: The values range is the content of the database; operations are functions to calculate the query results a set of operations for the relational model
- **Relational Calculus**: Descriptive approach that is based on mathematical logic
  - higher-level declarative language for specifying relational queries,

2.1 Relational Algebra

- e.g., no order of operations, only what information the result should contain

2.1 Relational Algebra

Operations

Operation	Purpose	Notation
Selection	Selects all tuples that satisfy the selection condition from a relation R	$\sigma_{<selection\ condition>}(R)$
Projection	Produces a new relation with only some of the attributes of R, and removes duplicate tuples	$\pi_{<attribute\ list>}(R)$
Renaming	Column in the result relation gets new name	$\rho_{new\ name \leftarrow attribute\ name}(R)$
Join	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition	$R_1 \bowtie_{<join\ condition>} R_2$
Equijoin	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons	$R_1 *_{<join\ condition>} R_2$

2.1 Relational Algebra

Operations

Operation	Purpose	Notation
Union	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible	$R_1 \cup R_2$
Intersection	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible	$R_1 \cap R_2$
Set Difference	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible	$R_1 - R_2$
Cartesian Product	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2	$R_1 \times R_2$

2.2 Subqueries

Basics

- `SELECT` returns relation: a (multi-)set
- Result of `SELECT` can be included in query
  - `WHERE` clause
    - also, for `UPDATE`, `DELETE`
  - `HAVING` clause
  - `FROM` clause
- `SELECT` clause (in column list)
- So, we have two (or more) `SELECT`s:
  - Outer `SELECT`
  - Nested (or inner) `SELECT`: **subquery**

2.2 Subqueries

Operations in `WHERE`

2.2 Subqueries

IN

```
SELECT E.Fname, E.Lname
FROM   EMPLOYEE AS E
WHERE  E.Ssn IN (   SELECT Essn
                    FROM   DEPENDENT AS D
                    WHERE  E.Sex = D.Sex );
```

=

```
SELECT *
FROM y
WHERE x = ( SELECT MAX(x) FROM y ) ;
```

ANY

```
SELECT name
FROM   Person
WHERE  PNr = ANY (SELECT PNr FROM book ) ;
```

ALL

```
SELECT Lname, Fname
FROM   EMPLOYEE
WHERE  Salary > ALL
       (SELECT       Salary
       FROM          EMPLOYEE
       WHERE         Dno=5) ;
```

EXISTS

```
SELECT E.Fname, E.Lname
FROM   EMPLOYEE AS E
WHERE  EXISTS
              (SELECT *
              FROM   DEPENDENT AS D
              WHERE  E.Ssn = D.Essn
                     AND E.Sex = D.Sex);
```

NOT EXISTS

```
SELECT Fname, Lname
FROM   EMPLOYEE
WHERE  NOT EXISTS ( SELECT *
                    FROM   DEPENDENT
                    WHERE  Ssn=Essn );
```

2.2 Subqueries

In `FROM`

- `SELECT` returns a new relation
- … so, we can select values from it
- Necessary: give a name to the relation
- Example: Alias name `newtab_b`

```
SELECT tab_a.x , newtab_b.y FROM tab_a , (SELECT v1, v2 FROM
tab_b) AS newtab_b;
```

2.2 Subqueries

Assignment

2.2 Subqueries

- Suppose that we want the department name number of employees in each department whose departments have an average income of more than 30,000 $. How can we specify this query in SQL?
- Retrieve the names of all employees with the smallest salary.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

2.2 Subqueries

- Retrieve the names of all employees whose supervisor's supervisor has ssn '888665555'.
- Retrieve the names of employees who make at least $10,000 more than the employee who is paid the least in the company.

2.3 Views

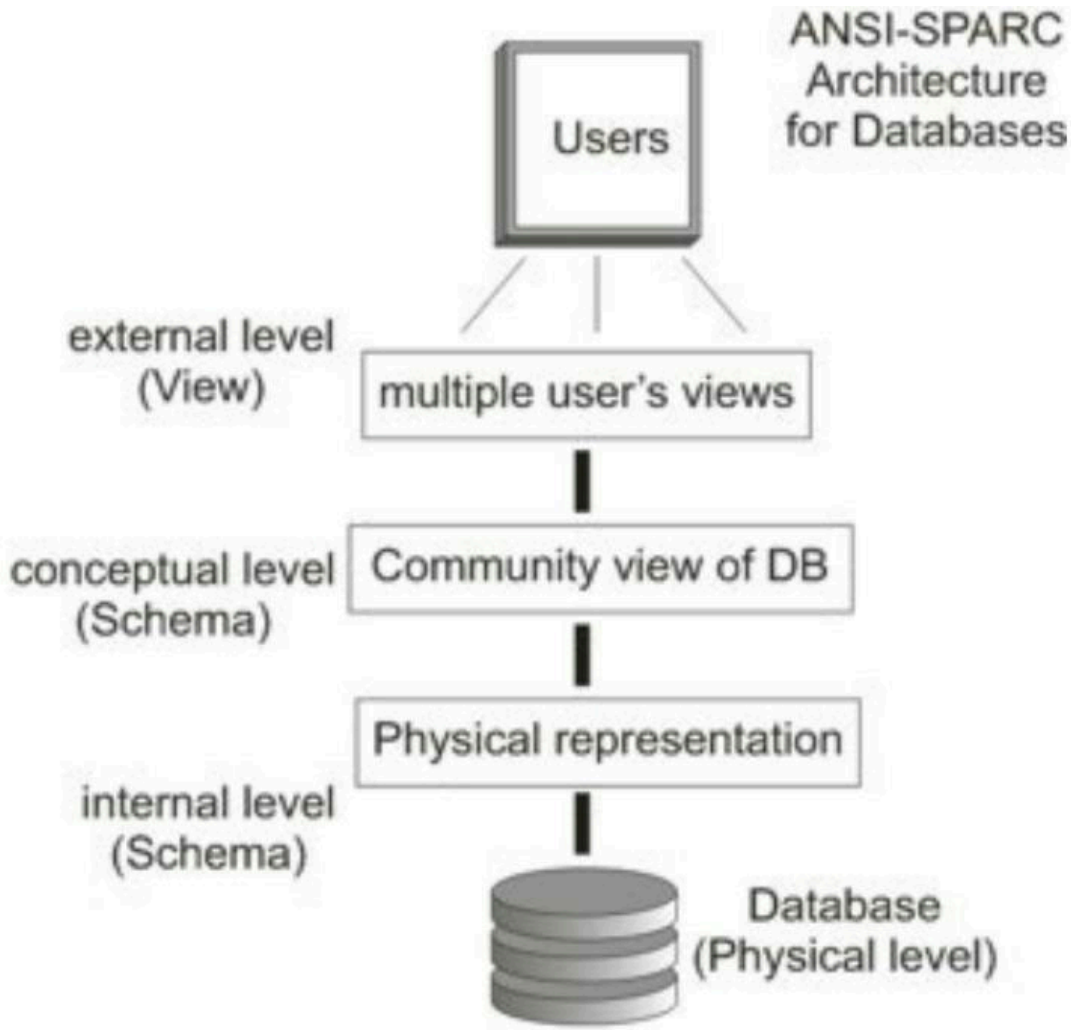
ANSI-SPARC

2.3 Views

External Schema

Conceptual Schema

Internal Schema



2.3 Views

Basics

- User or application specific views on data
- Only relevant portions of the data
- A **view** in SQL terminology is a single table that is derived from other tables Other tables can be **base tables** or previously defined views
- A view is considered to be a **virtual table**
  - In contrast to base tables
- Limits the possible update operations

2.3 Views

- No limitations on querying a view

2.3 Views

Example

```sql
CREATE VIEW vPerson AS
SELECT Name , Id , BirthDate FROM person;
```

Can rename columns in view:

```sql
CREATE VIEW vPerson (lname, pnr, bd) AS
SELECT Name , Id , BirthDate FROM person;
```

3. Views

3.1 Updating Views

Basics

- Views are Relations, just like tables!
- Should make no difference to users

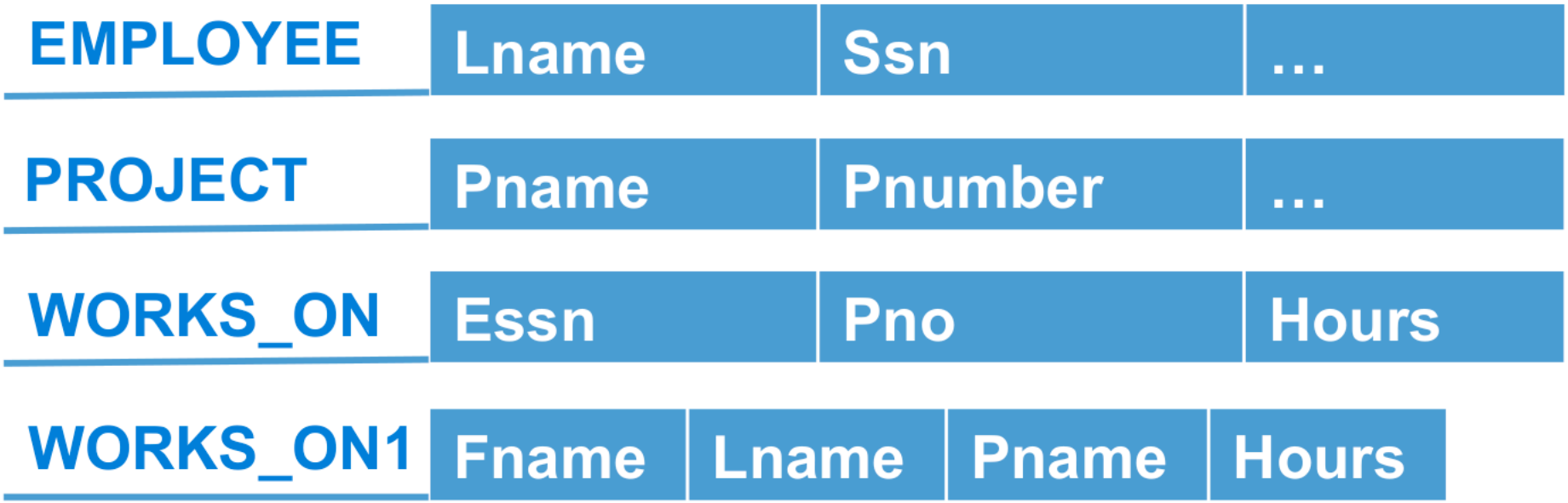
? Question

Can we modify the view's data?

- Depends on type of view!

3.1 Updating Views

Basics




```sql
UPDATE v_WORKS_ON1
SET Pname = "Project2"
```

3.1 Updating Views

```
3  WHERE Lname= "Borg"
4  AND Fname= "James"
5  AND Pname= "Project2";
```

3.1 Updating Views

Basics

- Possibility 1

```SQL
UPDATE WORKS_ON SET
Pno= SELECT Pnumber FROM PROJECT
WHERE Pname= "Project2")
WHERE Essn IN (SELECT Ssn FROM
EMPLOYEE WHERE Lname= "Borg" AND Fname= "James")
AND Pno = ( SELECT Pnumber FROM PROJECT WHERE Pname= "Project1");
```

3.1 Updating Views

Basics

- Possibility 2

```sql
UPDATE PROJECT
SET Pname = "Project2"
WHERE Pname = "Project1";
```

3.1 Updating Views

Basics

- Classify views based on the select:
  - **Projection View**
    - `SELECT a, b, c ...`
  - **Selection View**
    - `... WHERE <condition> ...`
  - **Join View**
    - `FROM tab_a JOIN tab_b ...`
    - **Aggregation View**
      - `SELECT MAX(x) ...`
- Other types and combinations exist

3.1 Updating Views

Basics

- Projection View

Example

```SQL
1  ... AS SELECT a , b , c FROM ...
```

- Manipulations can be transformed to base table quite easily
- Problems:
  - INSERT
    - NOT NULL columns in base table
  - DELETE

3.1 Updating Views

  - Problem if projection does not contain primary key
- In general: Can violate integrity constraints of base table

3.1 Updating Views

Basics

- **Selection View**

Example

```sql
CREATE VIEW v_top AS SELECT * FROM employee
WHERE salary > 20000;
```

SQL

- Problem:
  - Update can move tuples out of selection condition
  - So, the update looks like a delete!

3.1 Updating Views

Example

```sql
UPDATE v_top SET salary = 100;
```

- This phenomenon is called “**tuple migration**”

3.1 Updating Views

Basics

- **Join View**

Example

```sql
CREATE VIEW v_depman AS
    SELECT *
    FROM employee , department
    WHERE employee.ssn = department.mgr_ssn ;
```

3.1 Updating Views

- Data manipulation cannot be transformed to base tables in general case!

```sql
DELETE FROM v_depman WHERE id =11;
```

- Transformation to base tables employee and department?

```sql
DELETE FROM employee?
DELETE FROM department?
```

3.1 Updating Views

Basics

- **Aggregation View**

Example

```sql
CREATE VIEW v_astats AS
   SELECT MAX(i) , MIN(i) , COUNT(*)
   FROM a ;
```

- Update of the aggregated columns not possible!

3.1 Updating Views

> **Info**
>
> Aggregation may depend on other columns (`GROUP BY`)

3.1 Updating Views

Basics

- A view with a single defining table is updatable if
  - the view attributes contain the primary key of the base relation,
  - as well as all attributes with the NOT NULL constraint that have a default value specified
- Views defined on multiple tables using joins are only updatable in special cases
  - E.g., INSERT and UPDATE for Join Views, if join condition is based on PK-FK
- Views defined using grouping and aggregate functions are not updatable

3.1 Updating Views

Basics

- Oracle and standard SQL allow certain options at end of VIEW definition:
  - `WITH READ ONLY;`
    - Read only view, no data manipulation allowed
  - `WITH CHECK OPTION;`
    - Updates leading to tuple migration are denied

3.1 Updating Views

Generated Tables

```sql
CREATE TABLE <name> AS SELECT
```

SQL

- Can create new table based on query
- New table is independent from old table
- Use cases:
  - Copy table
  - Copy parts of table

3.1 Updating Views

! Memorize

New table does not have all constraints of the parent table!

3.1 Updating Views

Generated Tables

Example

```sql
INSERT INTO Underpaid ( lname , fname )
  SELECT lname , fname **FROM** Employee WHERE salary < 1000 ;
```

- `WHERE` clause belongs to `SELECT`

3.1 Updating Views

Views: Assignment

3.1 Updating Views

- Create a view that has the department name, manager name, and manager salary for every department.
- Create a view that has the project name, controlling department name, number of employees, and total hours worked per week on the project for each project.
- Create a view that has the project name, controlling department name, number of employees, and

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

3.1 Updating Views

total hours worked per week on the project for each project with more than one employee working on it.

3.1 Updating Views

Views: Assignment

3.1 Updating Views

Consider the following view `v_DEPT_SUMMARY` defined on the `COMPANY` database:

```sql
CREATE VIEW
v_DEPT_SUMMARY (DNO,
COUNT_EMPS,
SUM_SALARY,
AVG_SALARY) AS
SELECT Dno, COUNT(*),
SUM(Salary), AVG(Salary) FROM
EMPLOYEE GROUP BY Dno;
```

SQL

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

3.1 Updating Views

Views: Assignment

Describe the semantics oft he following SQL -statements. State which of the following queries and updates would be allowed on the view. If a query or update would be allowed, show what the corresponding query or update on the base relations would look like, and give its result when applied to the database.

```sql
SELECT * FROM v_DEPT_SUMMARY;
```

```sql
SELECT DNO, COUNT_EMPS FROM V_DEPT_SUMMARY WHERE
SUM_SALARY > 100000;
```

3.1 Updating Views

```
   SELECT DNO, AVG_SALARY FROM V_DEPT_SUMMARY WHERE
3  COUNT_EMPS > (SELECT COUNT_EMPS FROM V_DEPT_SUMMARY WHERE
   DNO=4);
4  UPDATE v_DEPT_SUMMARY SET DNO=3 WHERE DNO=4;
5  DELETE FROM v\_DEPT\_SUMMARY WHERE COUNT\_EMPS > 4;
```

3.1 Updating Views

Views: Assignment

3.1 Updating Views

```sql
1  CREATE VIEW
   v_DEPT_SUMMARY (DNO,
   COUNT_EMPS,
   SUM_SALARY,
   AVG_SALARY)
2  AS SELECT Dno, COUNT(*),
   SUM(Salary), AVG(Salary)
3  FROM EMPLOYEE GROUP BY Dno;
4
5  SELECT * FROM
   v_DEPT_SUMMARY;
```

SQL

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

3.1 Updating Views

```
6  SELECT DNO, COUNT_EMPS FROM
   v_DEPT_SUMMARY WHERE
   SUM_SALARY > 100000;
7  SELECT DNO, AVG_SALARY FROM
   v_DEPT_SUMMARY WHERE
   COUNT_EMPS >
8  (SELECT COUNT_EMPS FROM
   v_DEPT_SUMMARY WHERE DNO=4);
9
10 UPDATE v_DEPT_SUMMARY SET
   DNO=3 WHERE DNO=4;
```

3.1 Updating Views

```
11  DELETE FROM v_DEPT_SUMMARY
    WHERE COUNT_EMPS > 4;
```

4. Transactions

Operations

- A transaction bundles several operations into one logical unit
  - Unit of Work
- Includes one or more database access operations E.g., INSERT, DELETE, UPDATE, SELECT
- Operations must be executed all or none
- Example: Order a hotel room over the internet
  - Choose and reserve room
  - Payment
  - Final booking of the hotel room

Concurrency

- DBMS allow
  - many users &
  - concurrent access
- May lead to funny results if actions interfere

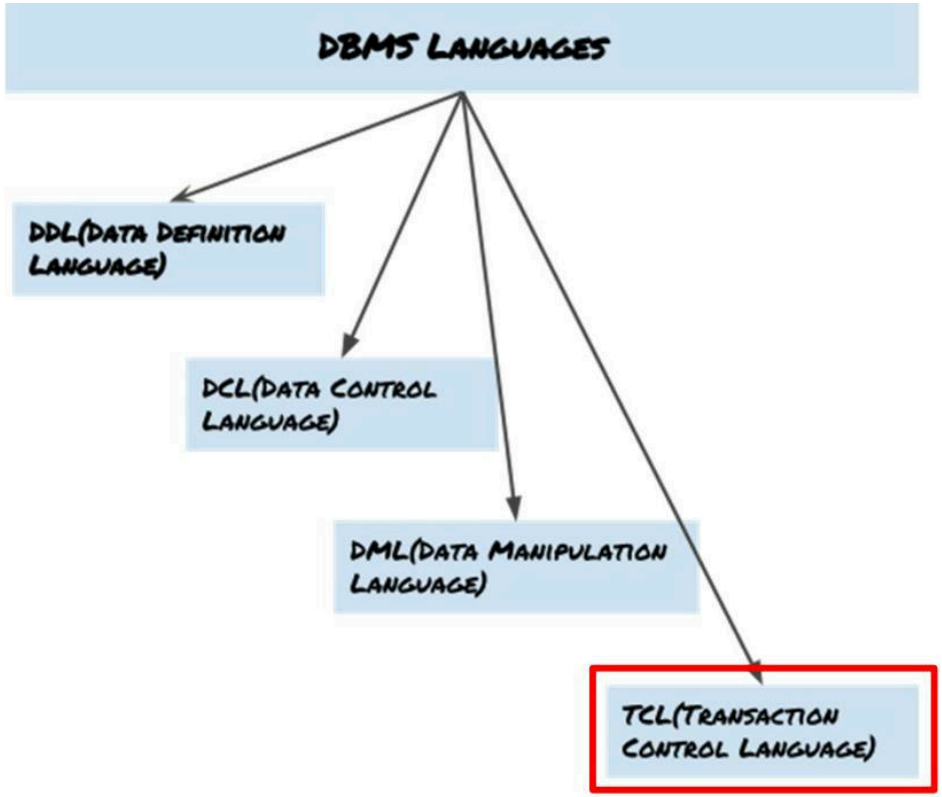
Example

Donald and Daisy withdraw money from their shared bank account

Languages

TCL for performing or rollbacking of changes in the database that we made using DML commands

- `BEGIN`
- `COMMIT` To persist the changes made by DML commands in database
- `ROLLBACK` To rollback the changes made to the database



4.1 Basics

- SAVEPOINTS

4.1 Basics

ACID

- Key features of transactions
  - **Atomicity**: Transaction is executed in whole or not at all
  - **Consistency**: State of the DB is consistent before and after a transaction
  - **Isolation**: Transactions do not interfere with other concurrent transactions
  - **Durability**: Changes are stored permanently in the database and will not get lost

ACID - Atomicity

- A transaction can consist of many operations
  - `SELECT`
- `INSERT`, `UPDATE`, `DELETE`
- Note: statements for data definition (e.g., `CREATE TABLE`) usually outside transaction!
- Single operations are always atomic
  - Not trivial when looking at the implementation!
- In a transaction, all operations or none are performed

ACID - Atomicity

- Begin of Transaction (**BoT**)
  - SQL99: START TRANSACTION
  - PostgresQL:

```sql
BEGIN;
```

- Commit a transaction: `COMMIT;`
  - All operations are made persistent
  - All changes are visible to other users
- Rollback transaction: `ROLLBACK;`
  - DB is in state at **BoT** again

ACID - Atomicity

- Autocommit
  - On some systems:
  - Single operations are committed automatically
  - Called `autocommit` mode
- May be turned off
  - … by disabling it
  - … by explicitly starting a transaction

> **Info**
>
> Method depends on system!

ACID - Consistency

- DB: in consistent state before transaction Also, in consistent state after transaction
- Integrity constraints assure that
- Constraints can be defined as
  - `IMMEDIATE` (default in mySQL)
    - are checked immediately after operation
  - `DEFERRED`
    - Check at time of commit

ACID - Consistency

Example

Employee ⟺ Department If the relationship “**employee works in department**” is mandatory, so that an employee should only exist, if he/she is working in a department, then how can that be assured?

- We must introduce a deferred constraint!

- PostgresQL: `SET CONSTRAINTS { ALL | name [, ...] } { DEFERRED | IMMEDIATE }`

4.1 Basics

Employee	eid
	1

Works_in	eid (FK)	did (FK)
	1	1

Department	did
	1

4.1 Basics

ACID - Isolation

- Transactions are isolated from other concurrent transactions
- Concurrent transactions shall behave well

ACID - Isolation: Concurrency Control

- Concurrent operations can lead to problems
  - Lost Update
  - Dirty Read
    - Unrepeatable read
    - Phantom tuples

ACID - Isolation: Concurrency Control

- Several transactions change the same value

Time	Transaction 1	Transaction 2
1	`SELECT price`	-
2	-	`SELECT price`
3	`UPDATE price = 5`	-
4	-	`UPDATE price = 6`
5	`COMMIT`	-
6	-	`COMMIT`

Article	Price
Pen	1
Eraser	2
Ruler	3

ACID - Isolation: Concurrency Control

- Transaction reads temporary value

Time	Transaction 1	Transaction 2
1	`SELECT price`	-
2	-	`SELECT price`
3	`UPDATE price = 7`	-
4		`SELECT price`
5	`ROLLBACK`	-

Article	Price
Pen	1
Eraser	2
Ruler	3

- A dirty read (also uncommitted dependency) occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed

ACID - Isolation: Concurrency Control

- Transaction receives inconsistent value due to interfering transaction
- Credit account and debit account have to match with their values!

Time	Transaction 1	Transaction 2
1	`SELECT amount`	-
2	-	`SELECT amount`
3	-	`INSERT new_amount`
4	-	`COMMIT`
5	`SELECT amount`	-
6	`COMMIT`	-

ACID - Isolation: Concurrency Control

4.1 Basics

- When doing the same `SELECT` twice, new tuples may appear that are inserted by another transaction
- Basically, same problem as Unrepeatable Read

Time	Transaction 1	Transaction 2
1	`SELECT credit_account`	-
2	-	`UPDATE credit_account`
3	-	`UPDATE debit_account`
4	-	`COMMIT`
5	`SELECT debit_account`	-
6	`COMMIT`	-

Article	Price
Pen	1
Eraser	2
Ruler	3

Article	Price
Pen	1
Eraser	2
Ruler	3
Ink	2

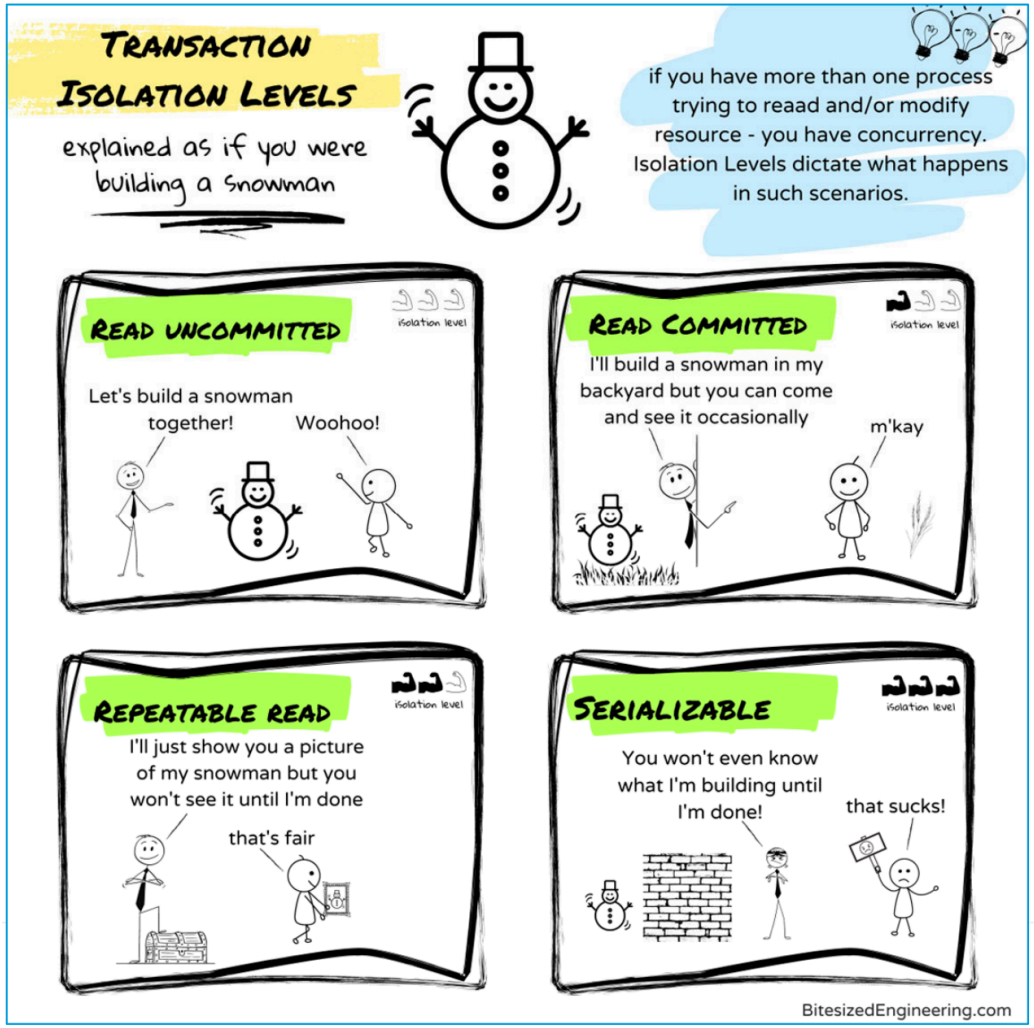
ACID - Isolation: Concurrency Control

- Lost Update is prevented by SQL
- Transactions: may choose **Isolation Level**
  - `SERIALIZABLE`
    - no problems
    - `REPEATABLE READ` (default in mySQL)
      - Open for phantom tuples
    - `READ COMMITTED` (default in Oracle, SQL Server)
      - Open for phantom tuples and unrepeatable read
    - `READ UNCOMMITTED`
      - Open for all problems

ACID - Isolation: Concurrency Control

Isolation level \ Read phenomena	Lost updates	Dirty reads	Non-repeatable reads	Phantoms
Read Uncommitted	may occur	may occur	may occur	may occur
Read Committed	may occur	don't occur	may occur	may occur
Repeatable Read	don't occur	don't occur	don't occur	may occur
Serializable	don't occur	don't occur	don't occur	don't occur

ACID - Isolation: Concurrency Control


TRANSACTION ISOLATION LEVELS
explained as if you were building a snowman
if you have more than one process trying to reaad and/or modify resource - you have concurrency. Isolation Levels dictate what happens in such scenarios.
READ UNCOMMITTED
isolation level
Let's build a snowman together!
Woohoo!
READ COMMITTED
isolation level
I'll build a snowman in my backyard but you can come and see it occasionally
m'kay
REPEATABLE READ
isolation level
I'll just show you a picture of my snowman but you won't see it until I'm done
that's fair
SERIALIZABLE
isolation level
You won't even know what I'm building until I'm done!
that sucks!
BitesizedEngineering.com

ACID - Isolation: Concurrency Control

- Isolation levels can be set
- Syntax:

```SQL
SET TRANSACTION < transaction_mode > [, …]
<transaction_mode> ::= ISOLATION LEVEL {
                            SERIALIZABLE |
                             REPEATABLE READS |
                             READ COMMIT |
                             READUNCOMMITED}
```

4.1 Basics

- Transactions can be **read only** if it contains only retrieval operations

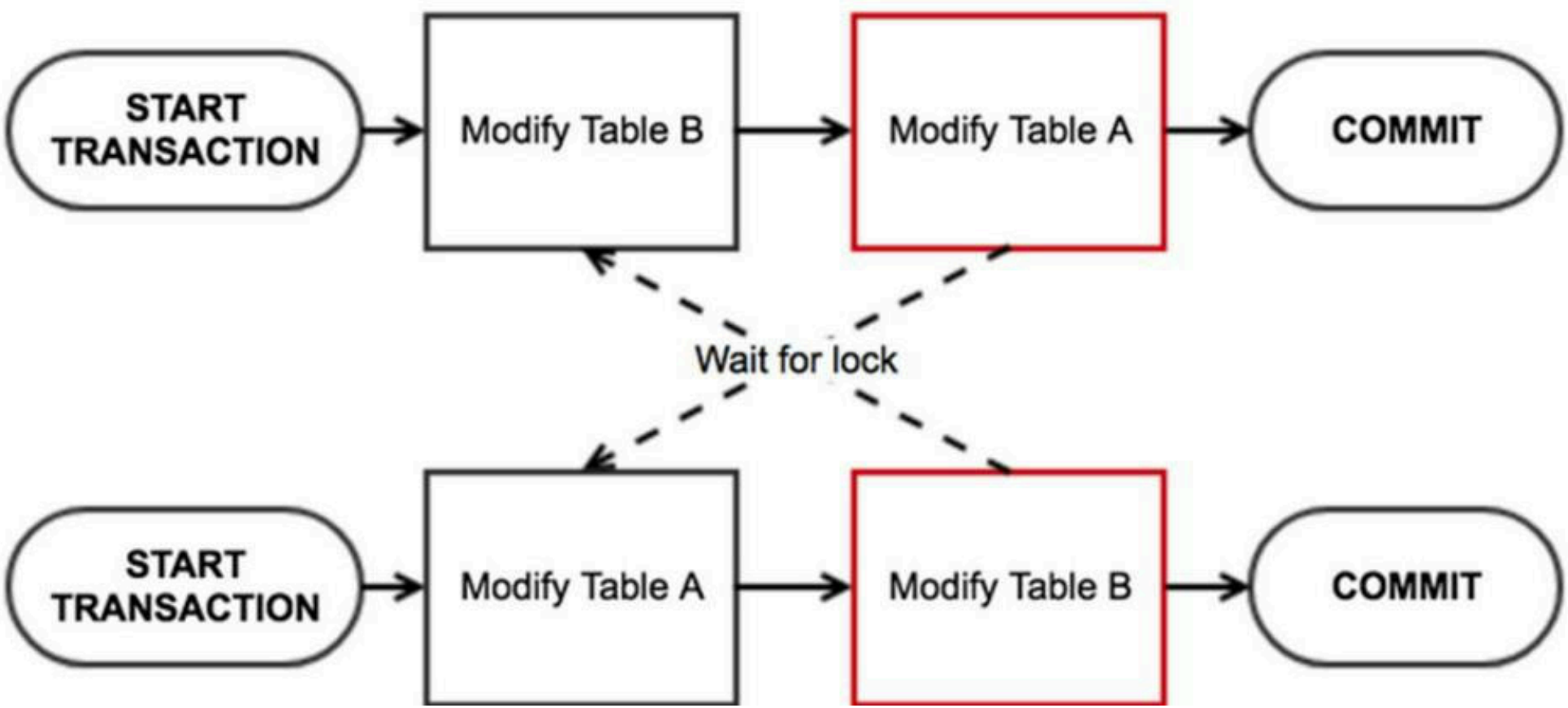
ACID - Isolation: Concurrency Control

- Locks
  - Read Lock (share)
  - Write Lock (exclusive)
- Locks may hold for
  - Row
  - Table
  - Also: Memory page, Disk block

ACID - Isolation: Concurrency Control

- Deadlocks may occur!
  - Usually are resolved automatically by aborting one transaction

4.1 Basics


START TRANSACTION
Modify Table B
Modify Table A
COMMIT
Wait for lock
START TRANSACTION
Modify Table A
Modify Table B
COMMIT

ACID - Isolation: Concurrency Control

- Multiversion Concurrency Control
  - Transactions see state of the database at **BoT** (begin of transaction)
  - Can reduce number of locks
  - DB has to store different versions of tuples

4.1 Basics

Time	Object1	Object2	Object3
0	"Foo" by T1	"Bar" by T1	
1	"Hello" by T2		
2		(deleted) by T3	"Foo-Bar" by T3

- Conceptually similar to Subversion or Git

ACID - Durability

- Once committed, changed data is safe
- Error types
  1. Computer failure
  2. Transaction or system error (constraint violation, $\frac{x}{0}$, blackout, system crash)
  3. Local Errors
  4. Concurrency control enforcement
  5. Disk error (harddisk broken)
  6. Physical problems and catastrophes (fire, earthquake, robbery, …)

ACID - Durability: Error Handling

- Recovery from transaction failures usually means that the database is **restored** to the most recent consistent state just before the time of failure
- Minor damages due to error types 1-4 from slide “ACID – Durability”
  - DBMS provides handling
    - Recovery strategy is to identify any changes that may cause an inconsistency in the database
      - Changes are first written to redo logs (files on disk)
      - Written to database files after commit

ACID - Durability: Error Handling

- Extensive damage due to error types 5-6 from slide “ACID – Durability”
  - Recovery handling restores a past copy of the database from archival storage
  - Reconstructs a more current state by redoing the operations
  - Last transactions are lost!
- Solution: Redundancy
  - RAID (**r** edundant **a** rray of **i** ndependent **d** isks)
    - Data Replication by DBMS

ACID - Durability: Error Handling

- Changes are performed on (replicated to) several database instances
- Master/Slave
  - Updates only on one instance (master)
  - Slave: Read only vs. Standby
- Multi-Master
  - Updates on different instances
  - Needs conflict resolution strategy

ACID - Durability: Error Handling

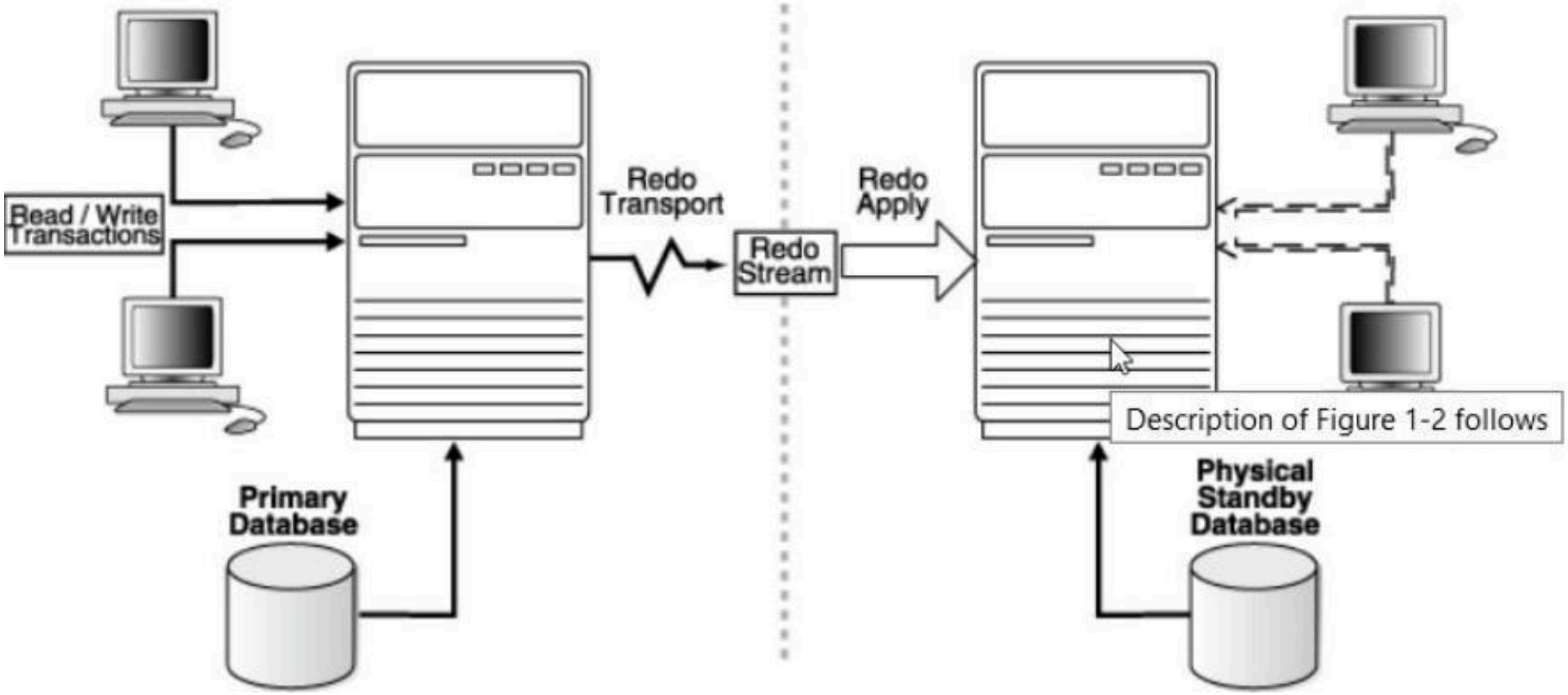
- **Synchronous**
  - Transaction valid only when committed on all DBs
  - Safest, but performance impact
  - May reduce availability of the system
- **Asynchronous**
  - Transaction valid when committed locally

ACID - Durability: Error Handling

- Low level (disk device)
- Trigger based
  - Update triggers the replication (SQL level)
- Logfile shipping
  - Changes are stored in redo logs (as usual)
  - redo logs are copied to standby DB

ACID - Durability: Error Handling

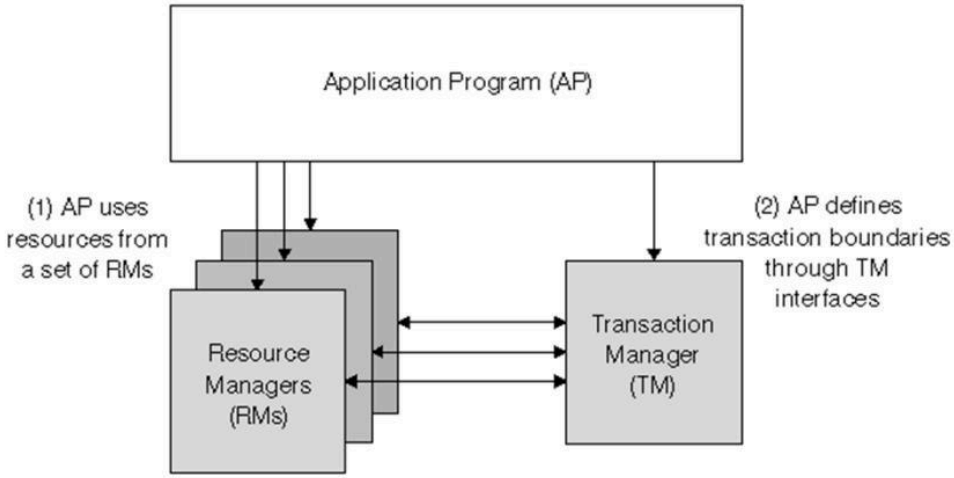
- Oracle
  - Data Guard
    - Replication on second server, can be used to answer Read-Only queries



- Real Application Cluster (RAC)
  - Several servers share the same DB

Distributed Transactions

- Transactions not only in a single DBS
- Standardized by X/Open
  - Transaction Manager: A software component that guarantees transaction properties
  - Resource Manager: Every resource (e.g., DBS, GUI) that is able to work in a



transactional mode without providing a transaction control structure itself

- The Transaction manager coordinates the Resource Manager that take part in the transaction. E.g., different DBS (distributed transactions) that appear as one DBS from outside (transparency!)

4.1 Basics

Distributed Transactions

Savepoints

- There are operations that may be expensive to execute time consuming
- If certain constraints fail within transaction execution, then maybe these constraints may not fail in a second attempt (e.g., time dependent)
- So “fall back” points can be defined, which are called **savepoints**
- It is possible to rollback up to a savepoint and restart transaction execution from this point on

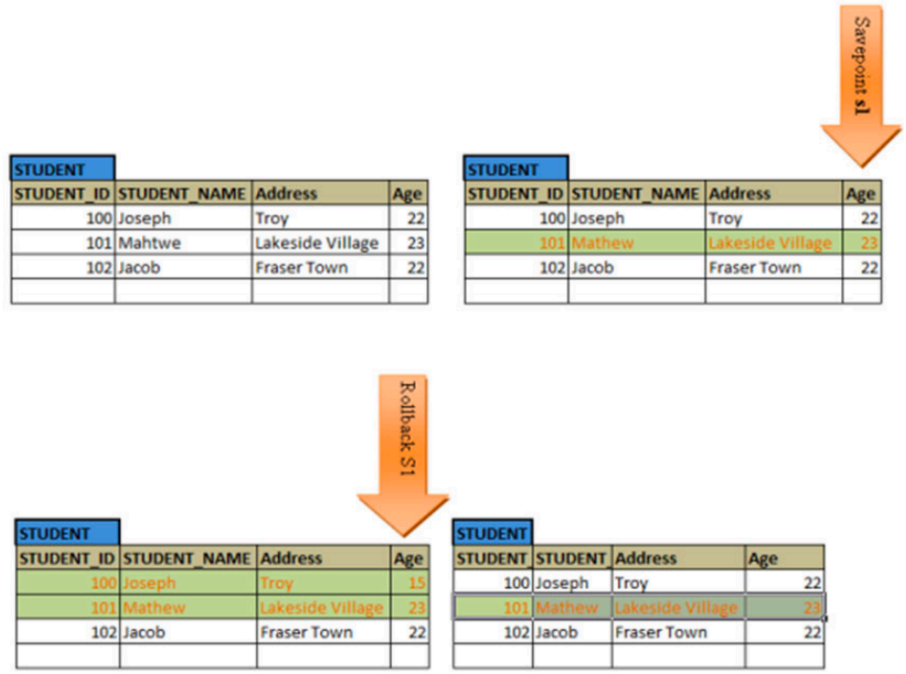
4.1 Basics

Savepoints

4.1 Basics

Code

```
UPDATE STUDENT SET STUDENT_NAME = 'Mathew' WHERE STUDENT_NAME = 'Mahtwe';
SAVEPOINT S1;
UPDATE STUDENT SET AGE = 15 WHERE STUDENT_ID = 100;
ROLLBACK to S1;
```



5. License Notice

5.1 Attribution

- This work is shared under the CC BY-NC-SA 4.0 License and the respective Public License.
- https://creativecommons.org/licenses/by-nc-sa/4.0/
- This work is based off of the work by Prof. Dr. Ulrike Herster.
- Some of the images and texts, as well as the layout were changed.
- The base material was supplied in private, therefore the link to the source cannot be shared with the audience.