Lecture 3 - BDA

1. Transactions
2. Normalization (1NF, 2NF, 3NF, BCNF)

Bonus: Security & Authorization

Transaction Management Overview

Chapter 16

ACID Properties

To preserve integrity of data, the database system must ensure:

- **Atomicity**. Either all operations of the transaction are properly reflected in the database or none are. **(All or Nothing)**
- **Consistency**. Execution of a transaction in isolation preserves the consistency of the database. **(No constraints violated)**
- **Isolation**. Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions. Intermediate transaction results must be hidden from other concurrently executed transactions. **(Users don’t affect each other's)**
- **Durability**. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. **(data committed, is permanent)**

Primitives Request

To preserve integrity of data, the database system must ensure:

- **! Any primitive request benefits from ACID properties:**
  i.e.: DELETE FROM Teacher WHERE age = 5
- Problem: granularity of these requests can be to thin.
- I.e.: Transfer from an account to another is 2 primitives.
  UPDATE Account SET total = total - 100 WHERE id = 123
  UPDATE Account SET total = total + 100 WHERE id = 124
- If there is a problem between the 2 requests, the database is in an inconsistent state.
- Solution: Put 2 requests in a “Transaction” which is a “super primitive”

Transactions

- A ***transaction*** is the DBMS's abstract view of a user program: a sequence of reads and writes.

- A user's program may carry out many operations on the data retrieved from the database, but the DBMS is only concerned about what data is read/written from/to the database.

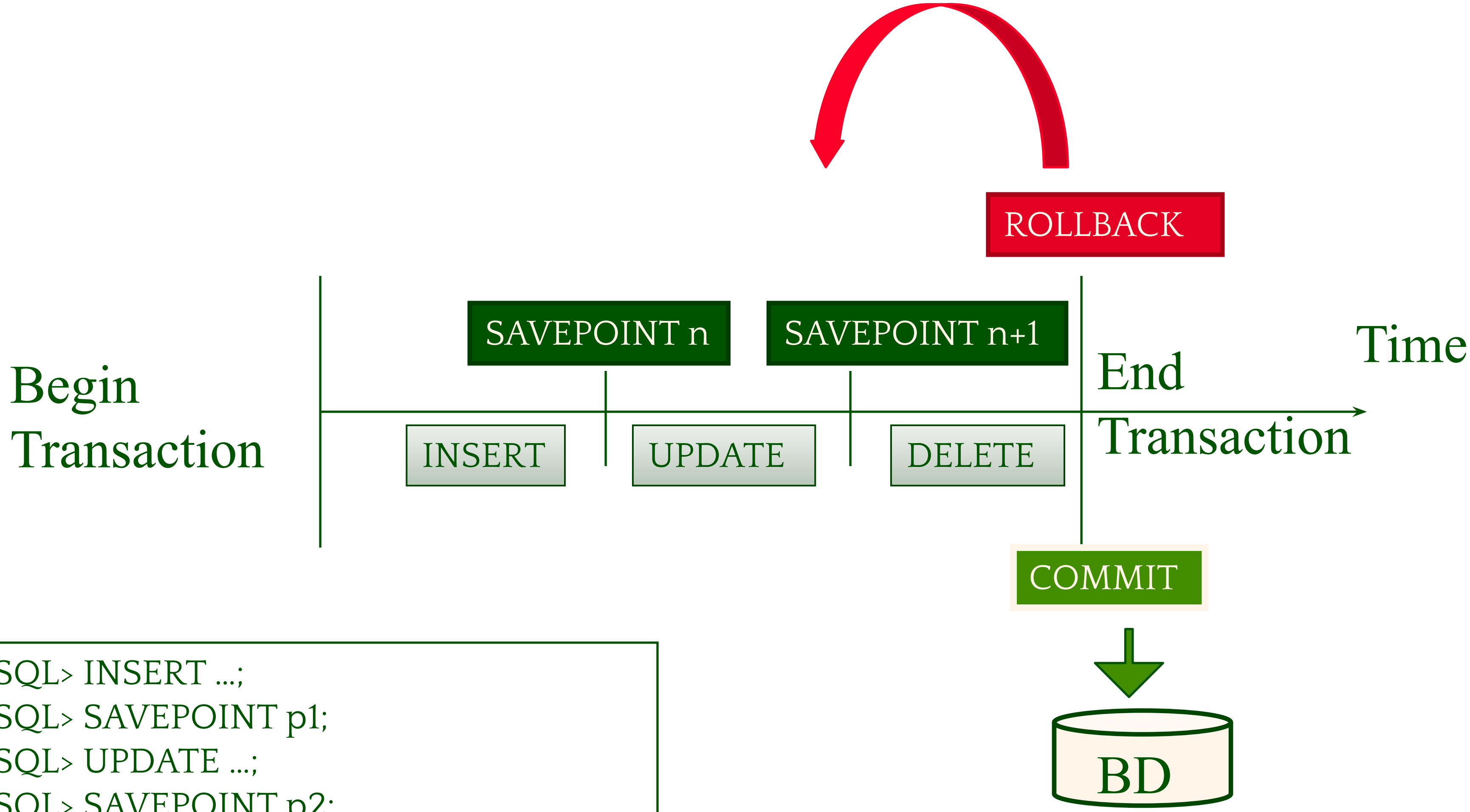
Transaction Concept

- A ***transaction*** is a *unit* of program execution that accesses and possibly updates various data items.
- A transaction must see a consistent database.
- During transaction execution the database may be inconsistent.
- When the transaction is committed, the database must be consistent.
- Two main issues to deal with:
  - Failures of various kinds, such as hardware failures and system crashes
  - Concurrent execution of multiple transactions

Atomicity of Transactions

- A transaction might *commit* after completing all its actions, or it could *abort* (or be aborted by the DBMS) after executing some actions.
- A very important property guaranteed by the DBMS for all transactions is that they are *atomic.* That is, a user can think of a transaction as always executing all its actions in one step, or not executing any actions at all.
  - DBMS *logs* all actions so that it can *undo* the actions of aborted transactions.
- show AUTOCOMMIT; (If true, all line are automatically committed. By default false)

Structuring the transaction




```
SQL> INSERT ...;
SQL> SAVEPOINT p1;
SQL> UPDATE ...;
SQL> SAVEPOINT p2;
SQL> UPDATE ...;
SQL> ROLLBACK TO p2;
```

Example

- Consider two transactions:

```
T1: BEGIN   A=A+100,   B=B-100   END
T2: BEGIN   A=1.06*A,   B=1.06*B   END
```

- Intuitively, the first transaction is transferring $100 from B's account to A's account. The second is crediting both accounts with a 6% interest payment.
- There is no guarantee that T1 will execute before T2 or vice-versa, if both are submitted together. However, the net effect *must* be equivalent to these two transactions running serially in some order.

Example (Contd.)

- Consider a possible interleaving (*schedule*):

T1:	A=A+100,		B=B-100
T2:		A=1.06*A,	B=1.06*B

- This is OK. But what about:

T1:	A=A+100,		B=B-100
T2:		A=1.06*A, B=1.06*B	

- A schedule of a set of transactions is a list of all actions where order of two actions from any transaction must match order in that transaction

T1:	R(A), W(A),		R(B), W(B)
T2:		R(A), W(A), R(B), W(B)	

Scheduling Transactions (Defs)

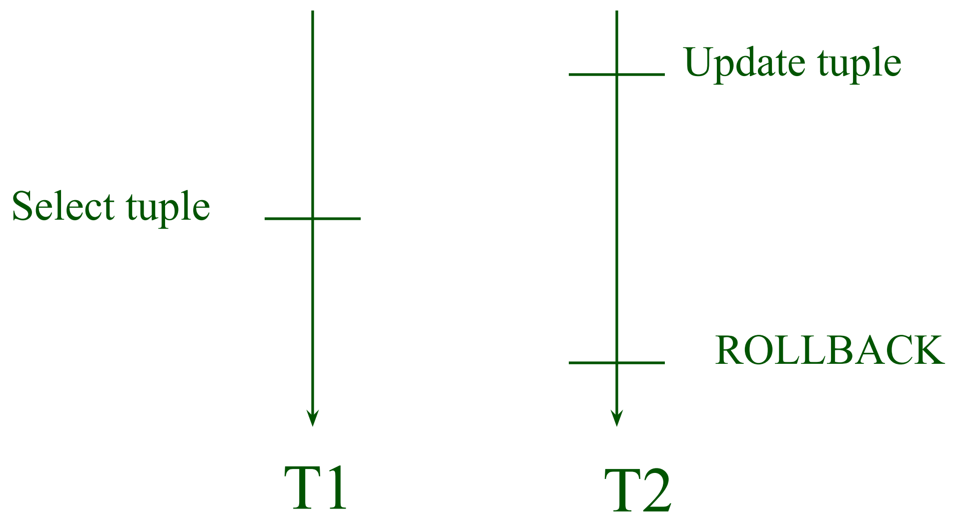
- *Serial schedule:* Schedule that does not interleave the actions of different transactions.
- *Equivalent schedules*: For any database state, the effect (on the set of objects in the database) of executing the first schedule is identical to the effect of executing the second schedule.
- *Serializable schedule*: A schedule which parallelize transactions and that is equivalent to some serial execution of the transactions.

(Note: If each transaction preserves consistency, every serializable schedule preserves consistency.)

Anomalies with Interleaved Execution

- Reading Uncommitted Data (WR Conflicts, "dirty reads"):

T1:	R(A), W(A), R(B), W(B), Abort
T2:	R(A), W(A), C



"dirty reads" Example

For example, we have two data A and B having to remain equal.

Transaction T1 (multiply by 2)	**Transaction T2 (adding 1)**
A = A*2	
	A = A+1
	B = B+1
B = B*2	

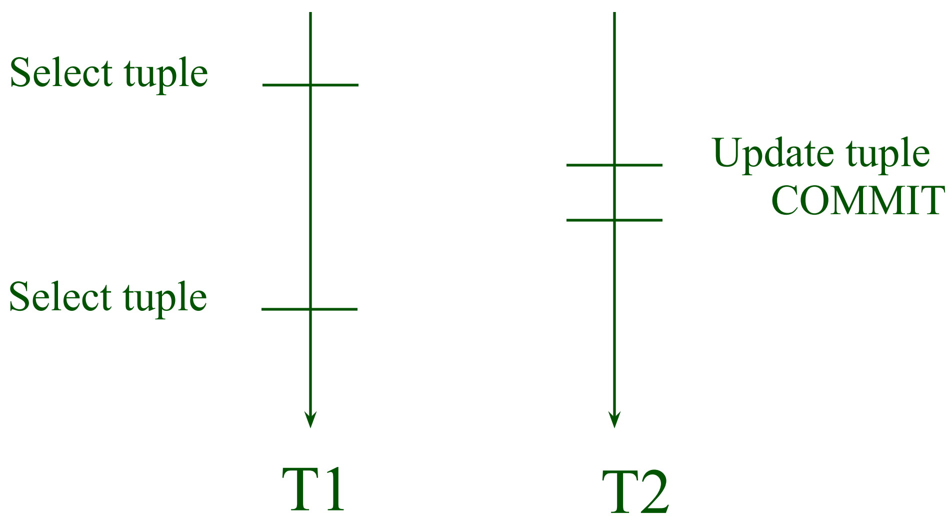
time

At the end of the two transactions the constraint A = B is not checked any more.

Anomalies with Interleaved Execution

- Unrepeatable Reads (RW Conflicts, “Fuzzy read”):

T1: R(A),	R(A), W(A), C
T2: R(A), W(A), C	

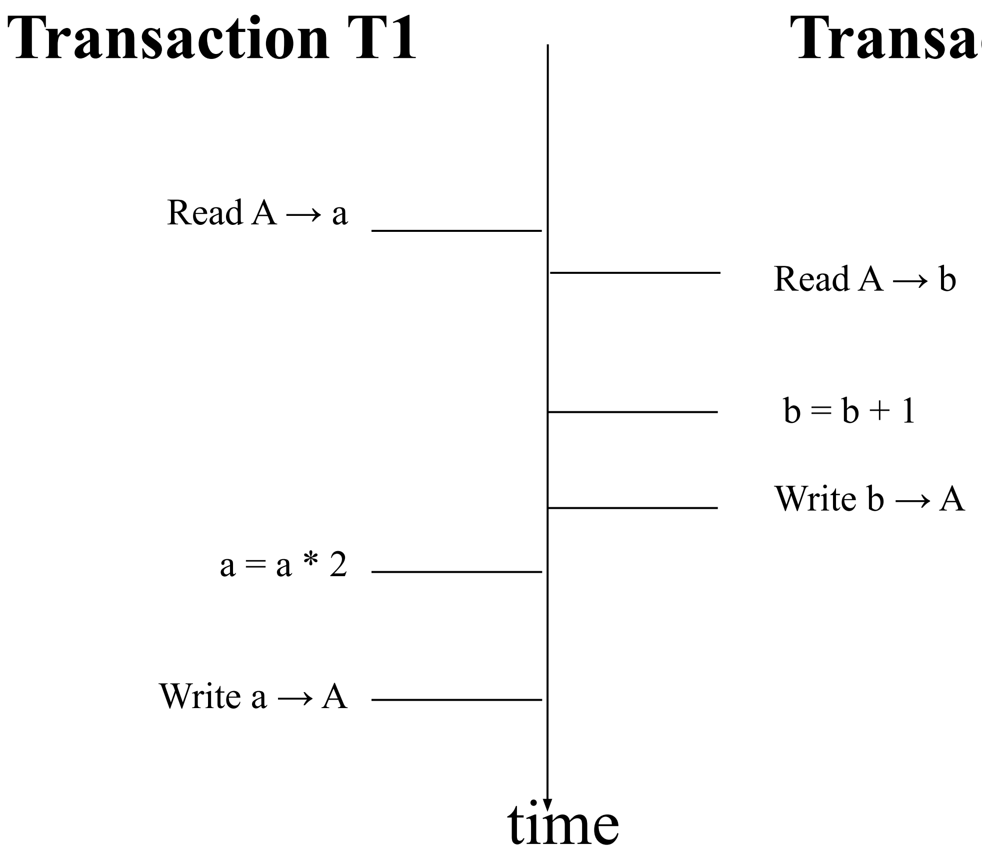


Anomalies (Continued)

- Overwriting Uncommitted Data (WW Conflicts): loss of update

T1:	W(A),		W(B), C	
T2:		W(A), W(B), C		

“Loss of update ” - example



Isolation

Isolation level	Dirty read	Fuzzy read	Loss of update
READ UNCOMMITED	YES	YES	YES
READ COMMITED *(default in postgres)*	✗	YES	YES
REPEATABLE READ	✗	✗	YES
SERIALIZABLE	✗	✗	✗

SQL> SET Transaction ISOLATION LEVEL READ COMMITTED;
SQL> ALTER SESSION SET ISOLATION_LEVEL=SERIALIZABLE

There are other problems to handle with concurrency, such as the 'Halloween' problem.

Isolation by database

- Most of the database uses "**READ COMMITED**" as a default isolation level:
  - Postgres
  - SqlServer
  - Oracle
  - SQLLite
- Some of them made it "**REPEATABLE READ**" as a default isolation level.
  - Mysql (therefore, MariaDb, Innodb ...)

Locks

- The DBMS ORACLE offers two types of lockings :
  - Locking on the level of the row (DML lock)
    - When a transaction modifies a table, only the rows to be modified are locked.
  - Locking on the level of the table (DDL lock)
    - When a transaction modifies a table, all the lines of the table are locked.

Consistency

- To avoid the transactions conflicts the DBMS must control the access to data.
- The technique most used to avoid the access conflicts to the data is the technique of prevention of the conflicts based on **locking**.

Locks

- Two types of locks :
  - Implicit : managed automatically by the DBMS
    - A transaction is opened implicitly
    - The COMMIT statement or ROLLBACK statement ends a transaction and opens a new one implicitly.
  - Explicit : specified by the user before begin a transaction

Locks

- There are several types of locks :
- <u>Lock in EXCLUSIVE Mode (X):</u>
  - With the order : LOCK TABLE table IN EXCLUSIVE MODE
  - This lock reserves all the lines of the table if no lock is already posed on the table or on one of the rows.
  - Only the current user can modify the contents of the table.
  - The other transactions can consult data of the table but can neither modify them nor to pose locks on the table or rows of the table.

Locks

- **LOCK in SHARE Mode (S) :**
  - With the order : LOCK TABLE table IN SHARE MODE
  - The user who posed the lock can consult and update the table if no other lock was posed. The other transactions can consult the data and pose various share locks on the table.
  - If several transactions lock a table in mode S, none of them can carry out an update on the data of the table.

Locks

- LOCK in ROW SHARE Mode (RS) :
  - With the order :

  LOCK TABLE table IN ROW SHARE MODE
  - It announces the intention to update a subset of lines of the table.
  - This mode prevents the blocking of the table in Exclusive mode.

Locks

- <u>LOCK in ROW EXCLUSIVE Mode (RX) :</u>
  - With the order :
    - LOCK TABLE table IN ROW EXCLUSIVE MODE
    - Or implicitly by Oracle when executing an UPDATE, INSERT or DELETE.

Locks

Example of implicit lock

UPDATE Shows SET Date ='01-02-05' WHERE NumShow =13 ;	
	UPDATE Shows SET Date ='02-02-05' WHERE NumShow = 13 ;
> 2 line(s) updated (lock lines of show N°13)	(stand by)
SELECT NumShow, Date FROM Shows; commit; <u>NumShow Date</u> 13 01/02/05	
	> 2 line(s) updated
	commit ;
SELECT NumShow, Date FROM Show ; <u>NumShow Date</u> 13 02/02/05	

Locks

- The deadlock
  - Deadlock occurs when

  a user awaits a resource taken by another user and that this other user awaits a resource taken by the first. The situation is then blocked.

- Example :
  - Transaction T1 modifies the date of the spectacles number 13 and 17. The T2 transaction modifies the hour of these spectacles in the following way :

Locks

Deadlock

UPDATE Show
SET Date ='01-02-05'
WHERE NumShow =13 ;

(lock lines of show N°13)

UPDATE Show
SET HourShow = '20-00'
WHERE NumShow = 17

(lock lines of show N°17)

UPDATE Show
SET Date ='01-02-05'
WHERE NumShow =17 ;

(wait)

UPDATE Show
SET HourShow = '20-00'
WHERE NumShow =13 ;

(wait)

Locks

- Resolution of the problem by ORACLE :
  - Automatic cancellation by implicit ROLLBACK of the most recent transaction.
- How to avoid the deadlock :
  - Deadlock would not have occurred if the two transactions update the table in the same order.
  - It is thus necessary always to reach the tables in the same order.
  - Any piece of identical treatment must be centralized in a block.
  - Avoid too long transactions.

Aborting a Transaction

.

- In order to *undo* the actions of an aborted transaction, the DBMS maintains a *log* in which every write is recorded. This mechanism is also used to recover from system crashes: all active transactions at the time of the crash are aborted when the system comes back up.

The Log

- The following actions are recorded in the log:
  - *Ti writes an object*: the old value and the new value.
  - *Ti commits/aborts*: a log record indicating this action.
- Log records are chained together by transaction id, so it's easy to undo a specific transaction.
- Log is often *archived* on stable storage.
- All log related activities (lock/unlock, dealing with deadlocks etc.) are handled transparently by the DBMS.

Schema Refinement and Normal Forms

Chapter 19

The Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - redundant storage, insert/delete/update anomalies
- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.
- Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).
- Decomposition should be used judiciously:
  - Is there reason to decompose a relation?
  - What problems (if any) does the decomposition cause?

Functional Dependencies (FDs)

- A functional dependency $X \rightarrow Y$ holds over relation R if, for every allowable instance *r* of R:
  - $t1 \in r,\ t2 \in r,\ \pi_X(t1) = \pi_X(t2)$ implies $\pi_Y(t1) = \pi_Y(t2)$
  - i.e., given two tuples in *r*, if the X values agree, then the Y values must also agree. (X and Y are *sets* of attributes.)
- K is a candidate key for R means that $K \rightarrow R$
  - However, $K \rightarrow R$ does not require K to be *minimal* !
- $X \rightarrow Y$
  - X determines Y
  - Y functionally depends on X
- I.e.: In this 2 tuples , namePlane -> numPlace, but not location
  - (namePlane = “747”, numPlace = “200”, location = “Paris”)
  - (namePlane = “747”, numPlace = “200”, location = “NYC”)

Example

Flight

NumFlight	NamePlane	Capacity	Location	NamePropr
100	A340	250	Toulouse	AF
101	A340	250	Toulouse	AF
102	A340	250	Paris	UA
103	B747	400	Paris	UA
104	B747	400	Paris	UA
105	Concorde	100	Paris	AF

Normal forms

- <u>Constraint</u> : all the airplanes which have the same name have the same capacity.
- <u>Problems</u> :
  - Logical redundancy : (A340, 250) appears several times in the table
  - Insertion anomaly : it is not possible to insert (B707, 150) because the key of the relation is NumAvion, which must be not null
  - Deletion anomaly it is not possible to remove the row (105, Concorde, 100, Paris, AF) because we lose information that Concorde has 100 seats.
  - Update anomaly if the capacity of an airplane increases must make the change on all tuples that contain the name of the aircraft.

36

Decomposition

- **Solution** : decompose the relation into « smaller » relations.
- **Définition** : A decomposition is a replacement of a relation R(A1,...,An) with a set of relations R1,...,Rm such that the join R1 $\bowtie \ldots \bowtie$ Rm has the same schema as the relation R

37

Decomposition 1

NumPlane	NomPlane	Capacity	Localisation	NamePropr
100	A340	250	Toulouse	AF
101	A340	250	Toulouse	AF
102	A340	250	Paris	UA
103	B747	400	Paris	UA
104	B747	400	Paris	UA
105	Concorde	100	Paris	AF

Flights	**NumFlight**	**NamePlane**	**Location**	**NameProp**
	100	A340	Toulouse	AF
	101	A340	Toulouse	AF
	102	A340	Paris	UA
	103	B747	Paris	UA
	104	A340	Paris	UA
	105	Concorde	Paris	AF

Planes	**NamePlane**	**Capacity**
	A340	250
	B747	400
	Concorde	100

Functional dependencies

- **Definition** : Suppose R(A1,...,An) is a relation, X and Y are two sets of A1,...,An.
- X determines Y or Y depends functionally on X ($X \rightarrow Y$) if, for every allowable instance *r* of R:
  - given two tuples in *r*, if the X values agree, then the Y values must also agree.

NamePlane → Capacity

NamePlane → NamePropr is not a FD

NamePlane → Location is not a FD

NumPlane	NomPlane	Capacity	Localisation	PlanePropr
100	A340	250	Toulouse	AF
101	A340	250	Toulouse	AF
102	A340	250	Paris	UA
103	B747	400	Paris	UA
104	B747	400	Paris	UA
105	Concorde	100	Paris	AF

Functional dependencies

- **Properties** :

1. **Reflexivity**: if $Y \subseteq X$ then $X \to Y$ (every set determines itself or part of itself)
2. **Augmentation** : if $X \to Y$ then $X Z \to Y Z$ (if X determines Y then the two sets can be augmented by a third set)
3. **Transitivity** : if $X \to Y$ and $Y \to Z$ then $X \to Z$
4. **Union** : if $X \to Y$ and $X \to Z$ then $X \to Y Z$
5. **Pseudo-transitivity** : if $X \to Y$ and $WY \to Z$ then $WX \to Z$
6. **Decomposition** : if $X \to Y$ and $Z \subseteq Y$ then $X \to Z$

Note : Properties 4 - 6 can be derived from properties 1- 3.

Functional dependencies

- <u>Definition</u> : An **elementary functional dependency** is a functional dependency $X \rightarrow A$ where there is not $X' \subset X$ such as $X' \rightarrow A$.
- <u>Exemple</u> : Suppose we have the following functional dependencies

  $F = \{ AC \rightarrow B, B \rightarrow CD, E \rightarrow BF, AE \rightarrow D \}$.

  - Dependencies B -> CD, E -> BF are not simple because their right side does not contain a single attribute.
  - Dependency AE -> D is not elementary because the dependency E -> D can be deduced from E -> BF and B -> CD.

Functional dependencies

- **Definition** : A set F° of FDs is a **minimum closure** of F if it satisfies the following properties:
  1. No dependencies is redundant in F °
  2. Any elementary FD from F ° is in the transitive closure F +
- The minimum closure can generate all the FDs.
- **Algorithm for computing the minimum closure of a set F of dependencies:**

```
G:=D; // D set of FDs
For each f ∈ G do
    If G – {f} implies f then
        G := G – {f} ;
    End If;
End For;
D° := G;
```

Functional dependencies

- Example: F is a set of functional dependencies

$$F = \{ AC \rightarrow B,\ B \rightarrow C,\ B \rightarrow D,\ E \rightarrow B,\ E \rightarrow F,\ E \rightarrow D \}.$$

- The minimum closure of F

$$F^\circ = \{ AC \rightarrow B,\ B \rightarrow C,\ B \rightarrow D,\quad E \rightarrow B,\ E \rightarrow F \}.$$

(because $E \rightarrow B$ and $B \rightarrow D$)

Functional dependencies

- **Definition** : A key of a relation R (A1, ..., An) is a subset X of attributes A1, ..., An such that:
  1. X => A1, ..., An
  2. There is no subset X' ⊂ X such that X' => A1, ..., An

1 NF

- First normal form:
  A relation is in first normal form (1NF) if any attribute value contains a single value (atomic).

Flights	**Flight**	**Day**	**Airport**	**Pilot**	**Qualif**
	SN890	Monday	Geneva	Alpha	1PL
		Thursday	Geneva	Bravo	2PL
	SN891	Monday	Brussel		
	SN891	Thursday	Brussel		
				Bravo	2PL

Not in 1 NF

Flights	**Flight**	**Day**	**Airport**	**Pilot**	**Qualif**
	SN890	Monday	Geneva	Alpha	1PL
	SN890	Thursday	Geneva	Bravo	2PL
	SN891	Monday	Brussel	Alpha	1PL
	SN891	Thursday	Brussel	Delta	1PL
	SN836	Wednesday	Geneva	Bravo	2PL

1 NF (Exercise)

- First normal form:
  A relation is in first normal form (1NF) if any attribute value contains a single value (atomic).

- What's wrong in next relation and transform it in 1NF

Customer

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659 555-776-4100
789	Maria	Fernandez	555-808-9633

1 NF (Solution)

Customer

~~Customer ID~~	~~First Name~~	~~Surname~~	~~Telephone Number~~
~~123~~	~~Robert~~	~~Ingram~~	~~555-861-2025~~
~~456~~	~~Jane~~	~~Wright~~	~~555-403-1659~~ ~~555-776-4100~~
~~789~~	~~Maria~~	~~Fernandez~~	~~555-808-9633~~

Not in 1 NF

Customer

Customer ID	First Name	Surname	Telephone Number
123	Robert	Ingram	555-861-2025
456	Jane	Wright	555-403-1659
456	Jane	Wright	555-776-4100
789	Maria	Fernandez	555-808-9633

2 NF

- **Second normal form:**
  A relation is in second normal form (2NF) if and only if:
  1. It is in 1NF
  2. Any attribute not belonging to a key does not depend on a portion of a key.

FLIGHT → AIRPORT

Flights	Flight	Day	Airport	Pilot	Qualif
	SN890	Monday	Geneva	Alpha	1PL
	SN890	Thursday	Geneva	Bravo	2PL
	SN891	Monday	Brussel	Alpha	1PL
	SN891	Thursday	Brussel	Delta	1PL
	SN836	Wednesday	Geneva	Bravo	2PL

Not in 2 NF

PlaneLocation	Flight	Airport
	SN890	Geneva
	SN891	Brussel
	SN836	Geneva

PlanePilot	Flight	Day	Pilot	Qualif
	SN890	Monday	Alpha	1PL
	SN890	Thursday	Bravo	2PL
	SN891	Monday	Alpha	1PL
	SN891	Thursday	Delta	1PL
	SN836	Wednesday	Bravo	2PL

2 NF (Exercise)

- **<u>Second normal form:</u>**
  A relation is in second normal form (2NF) if and only if:
  1. It is in 1NF
  2. Any attribute not belonging to a key does not depend on a portion of a key.

- What's wrong in next relation an how to transform it in 2NF ?

Employees' Skills

<u>Employee</u>	<u>Skill</u>	Current Work Location
Brown	Light Cleaning	73 Industrial Way
Brown	Typing	73 Industrial Way
Harrison	Light Cleaning	73 Industrial Way
Jones	Shorthand	114 Main Street
Jones	Typing	114 Main Street
Jones	Whittling	114 Main Street

2 NF (Solution)

Employee → work location

Employees' Skills

~~Employee~~	~~Skill~~	~~Current Work Location~~
~~Brown~~	~~Light Cleaning~~	~~73 Industrial Way~~
~~Brown~~	~~Typing~~	~~73 Industrial Way~~
~~Harrison~~	~~Light Cleaning~~	~~73 Industrial Way~~
~~Jones~~	~~Shorthand~~	~~114 Main Street~~
~~Jones~~	~~Typing~~	~~114 Main Street~~
~~Jones~~	~~Whittling~~	~~114 Main Street~~

Not in 2 NF

Employees

Employee	Current Work Location
Brown	73 Industrial Way
Harrison	73 Industrial Way
Jones	114 Main Street

Employees' Skills

Employee	Skill
Brown	Light Cleaning
Brown	Typing
Harrison	Light Cleaning
Jones	Shorthand
Jones	Typing
Jones	Whittling

3 NF

❖ **Third normal form:**
A relation is in third normal form (3NF) if and only if:
1. It is in 2NF
2. Any attribute not belonging to a key does not depend on another non-key attribute.

Flights	Flight	Day	Airport	Pilot	Qualif
	SN890	Monday	Geneva	Alpha	1PL
	SN890	Thursday	Geneva	Bravo	2PL
	SN891	Monday	Brussel	Alpha	1PL
	SN891	Thursday	Brussel	Delta	1PL
	SN836	Wednesday	Geneva	Bravo	2PL

PILOTE → QUALIF

Not in 3 NF

Pilots	Pilot	Qualif
	Alpha	1PL
	Bravo	2PL
	Delta	1PL

Flights	Flight	Day	Airport	Pilot
	SN890	Monday	Geneva	Alpha
	SN890	Thursday	Geneva	Bravo
	SN891	Monday	Brussel	Alpha
	SN891	Thursday	Brussel	Delta
	SN836	Wednesday	Geneva	Bravo

3 NF (Exercise)

- **Third normal form:**
  A relation is in third normal form (3NF) if and only if:
  1. It is in 2NF
  2. Any attribute not belonging to a key does not depend on another non-key attribute.

- What's wrong with next relation and how to transform it in 3NF ?

Tournament Winners

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

3 NF (Solution)

Winner => Winner Date of Birth

Tournament Winners

Tournament	Year	Winner	Winner Date of Birth
Indiana Invitational	1998	Al Fredrickson	21 July 1975
Cleveland Open	1999	Bob Albertson	28 September 1968
Des Moines Masters	1999	Al Fredrickson	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 March 1977

Not in 3 NF

Tournament Winners

Tournament	Year	Winner
Indiana Invitational	1998	Al Fredrickson
Cleveland Open	1999	Bob Albertson
Des Moines Masters	1999	Al Fredrickson
Indiana Invitational	1999	Chip Masterson

Winner Dates of Birth

Winner	Date of Birth
Chip Masterson	14 March 1977
Al Fredrickson	21 July 1975
Bob Albertson	28 September 1968

Summary and More

	UNF (1970)	1NF (1970)	2NF (1971)	3NF (1971)	EKNF (1982)	BCNF (1974)	4NF (1977)	ETNF (2012)	5NF (1979)	DKNF (1981)	6NF (2003)
Primary key (no duplicate tuples)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
No repeating groups	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Atomic columns (cells have single value)[8]	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either does not begin with a proper subset of a candidate key or ends with a prime attribute (no partial functional dependencies of non-prime attributes on candidate keys)[8]	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with a prime attribute (no transitive functional dependencies of non-prime attributes on candidate keys)[8]	✗	✗	✗	✓	✓	✓	✓	✓	✓	✓	✓
Every non-trivial functional dependency either begins with a superkey or ends with an elementary prime attribute[8]	✗	✗	✗	✗	✓	✓	✓	✓	✓	✓	N/A
Every non-trivial functional dependency begins with a superkey[8]	✗	✗	✗	✗	✗	✓	✓	✓	✓	✓	N/A
Every non-trivial multivalued dependency begins with a superkey[8]	✗	✗	✗	✗	✗	✗	✓	✓	✓	✓	N/A
Every join dependency has a superkey component[9]	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓	N/A
Every join dependency has only superkey components[8]	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	N/A
Every constraint is a consequence of domain constraints and key constraints[8]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Every join dependency is trivial[8]	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓

Relation decomposition

- <u>3NF decomposition algorithm</u>

Consider a relation R (A1, ..., An) with a set F of FDs.

<u>Step 1</u> : find the minimum closure F ° of F

<u>Step 2</u> : decomposition

- If there are attributes $A_{k1},\ldots,A_{ki}$ which do not appear in F°, then create a relation R_1 ($A_{k1},\ldots, A_{ki}$)..
- If there is a dependency that contains all the attributes of R then R is the result.
- Otherwise for each dependency $X_i \rightarrow A_i$ of the minimum closure of F we create a relation Ri (X_i ,A_i) . If there are multiple FDs $X_i \rightarrow A_{i1}$, ..., $X_i \rightarrow A_{ij}$ we obtain the relation Ri (X_i ,A_{i1}, ..., A_{ij}).

Relation decomposition

- **Example** : 3NF decomposition
- let R = (C, P, H, S, R, G) where :

C = Course	P = Professor
H = Hours	R = Room
S = Student	G = Grade

- **Functional dependencies**
  - $C \rightarrow P$: Each course is made by a single professor
  - $HR \rightarrow PC$: in a room at a fixed hour, there is one course with one professor
  - $HP \rightarrow R$: a professor can be in one room at a time
  - $CS \rightarrow G$: a student has a single grade for each course
  - $SH \rightarrow R$: a student can be in one room at a time

Relation decomposition

- To decompose R in 3NF, we compute the minimum closure of FDs

1) The dependency HR $\rightarrow$ PC does not contain a single attribute to the right. It is replaced by two FDs: HR $\rightarrow$ P and HR $\rightarrow$ C.
2) The dependency HR $\rightarrow$ P is redundant because it can be deduced from
   HR $\rightarrow$ C et C $\rightarrow$ P.

   The other dependencies are not redundant and can not be minimized to the left.

   The minimum closure is:

   { C $\rightarrow$ P, HR $\rightarrow$ C, HP $\rightarrow$ R ,CS $\rightarrow$ G, SH $\rightarrow$ R }.

Relation decomposition

- **Decomposition of R into 3 NF:**
  - R1 (C, P)
  - R2 (H, R, C)
  - R3 (H, P, R)
  - R4 (C, S, G)
  - R5 (S, H, R).

Boyce Codd

- If a relational schema is in BCNF then all redundancy based on functional dependency has been removed, although other types of redundancy may still exist. A relational schema R is in Boyce–Codd normal form if and only if for every one of its dependencies $X \rightarrow Y$, at least one of the following conditions hold:
  - $X \rightarrow Y$ is a trivial functional dependency ($Y \subseteq X$)
  - X is a super key for schema R

Boyce-Codd NF (example impossible)

- Before Decomposition:

ZipCode	<u>Street</u>	<u>City</u>
NW51AA	rue Londres	Londres
75006	rue NDC	Paris
75008	rue Londres	Paris
75001	rue de Rivoli	Paris

- (City, Street → ZipCode)
- (ZipCode → City).

2 streets with same name in different city is OK.

- (ZipCode → City). Is a BCNF violation
- So we should try decomposition:

  (ZipCode , City).

  (Street, ZipCode)

  But then we cannot ensure (City, Street → ZipCode)

Boyce-Codd NF (Achievability)

- Beeri and Bernstein showed in 1979 that, for example, a set of functional dependencies $\{AB \rightarrow C, C \rightarrow B\}$ cannot be represented by a BCNF schema.
- Thus, unlike the first three normal forms, BCNF is not always achievable.

Security and Authorization

Chapter 21

Authorization

- <u>A user is defined by</u> :
  - Name
  - password
  - Set of authorizations (privileges)
- Forms of authorization on parts of the database:
  - **Read authorization** - allows reading, but not modification of data.
  - **Insert authorization** - allows insertion of new data, but not modification of existing data.
  - **Update authorization** - allows modification, but not deletion of data.
  - **Delete authorization** - allows deletion of data

Authorization (Cont.)

Forms of authorization to modify the database schema:

- **Index authorization** - allows creation and deletion of indices.
- **Resources authorization** - allows creation of new relations.
- **Alteration authorization** - allows addition or deletion of attributes in a relation.
- **Drop authorization** - allows deletion of relations.

Security Specification in SQL

- The grant statement is used to confer authorization

  grant <privilege list>

  on <relation name or view name>

  to <user list>
- <user list> is:
  - a user-id
  - *public*, which allows all valid users the privilege granted
  - A role (more on this later)
- Granting a privilege on a view does not imply granting any privileges on the underlying relations.
- The grantor of the privilege must already hold the privilege on the specified item (or be the database administrator).

Privileges in SQL

- **select:** allows read access to relation,or the ability to query using the view
  - Example: grant users U_1, U_2, and U_3 **select** authorization on the *branch* relation:

    grant select on *branch* **to** U_1, U_2, U_3
- **insert**: the ability to insert tuples
- **update**: the ability to update using the SQL update statement
- **delete**: the ability to delete tuples.
- **references**: ability to declare foreign keys when creating relations.
- **all privileges**: used as a short form for all the allowable privileges

Privilege To Grant Privileges

- **with grant option**: allows a user who is granted a privilege to pass the privilege on to other users.
  - Example:

    grant select on *sailor* **to** U_1 **with grant option**

    gives U_1 the **select** privileges on branch and allows U_1 to grant this privilege to others

Roles

- Roles permit common privileges for a class of users can be specified just once by creating a corresponding "role"
- Privileges can be granted to or revoked from roles, just like user
- Roles can be assigned to users, and even to other roles
- SQL:1999 supports roles

create role *teller*
create role *manager*

grant select on *branch* **to** *teller*
grant update (*balance***) on** *account* **to** *teller*
grant all privileges on *account* **to** *manager*

grant *teller* **to** *manager*

grant *teller* **to** *alice, bob*
grant *manager* **to** *avi*

Revoking Authorization in SQL

- The **revoke** statement is used to revoke authorization.

  revoke<privilege list>

  on <relation name or view name> **from** <user list> [**restrict**|**cascade**]
- Example:

  revoke select on *branch* **from** U_1, U_2, U_3 **cascade**
- Revocation of a privilege from a user may cause other users also to lose that privilege; referred to as cascading of the **revoke**.
- We can prevent cascading by specifying **restrict**:

  revoke select on *branch* **from** U_1, U_2, U_3 **restrict**

  With **restrict**, the **revoke** command fails if cascading revokes are required.

Revoking Authorization in SQL (Cont.)

- <privilege-list> may be **all to** revoke all privileges the revokee may hold.
- If <revokee-list> includes **public** all users lose the privilege except those granted it explicitly.
- If the same privilege was granted twice to the same user by different grantees, the user may retain the privilege after the revocation.
- All privileges that depend on the privilege being revoked are also revoked.

GRANT/REVOKE on Views

- If the creator of a view loses the SELECT privilege on an underlying table, the view is dropped!
- If the creator of a view loses a privilege held with the grant option on an underlying table, (s)he loses the privilege on the view as well;

Views and Security

- Views can be used to present necessary information (or a summary), while hiding details in underlying relation(s).
  - Given ActiveSailors, but not Sailors or Reserves, we can find sailors who have a reservation, but not the *bid*'s of boats that have been reserved.
- Creator of view has a privilege on the view if (s)he has the privilege on all underlying tables.
- Together with GRANT/REVOKE commands, views are a very powerful access control tool.