



Day 24





Indexes

- Structure that improves data retrieval
- Separate table to store a sorted values of one or more columns
- Index is created automatically for primary key
 - Other columns are optional
- Indexes are useful for
 - Reducing query time
 - Grouping
 - Sorting
 - Minimum and maximum values



Index

Primary key (`prog_id`) index

Values are sorted

<code>idx_prog_id(prog_id)</code>
2
3
7
10

<code>prog_id</code>	<code>name</code>	<code>release_date</code>
10	Game of Thrones	2011-04-17
2	Get Smart	1965-09-18
7	Twin Peaks	1990-04-08
3	Hinterland	2013-10-29

`select * from tv_shows where prog_id = 3;` An indexed column

Search the index of `prog_id`. Since the values are sorted, only 2 entries are search.
If we have to search the `prog_id` from `tv_show` table, we will have to search 4 entries to get the result



Creating Index

- Creating an index

```
create index idx_name  
on tv_shows (name)
```

Index's name

For this column

tv_shows

prog_id	INT(11)
name	CHAR(64)
lang	CHAR(16)
official_site	VARCHAR(256)
rating	ENUM(...)
user_rating	FLOAT
release_date	DATE
image	BLOB

Indexes

- Speed up all queries where the predicate involves the name column only

```
select * from tv_shows where name like 'Game%';
```

- Index will slow down inserts
 - Have to insert into the index at the correct (sorted) location



Multi Column Index

```
create index lang_release_date_idx
on tv_shows (release_date, lang);
```

tv_shows	
🔑	prog_id INT(11)
💎	name CHAR(64)
💎	lang CHAR(16)
💎	official_site VARCHAR(256)
💎	rating ENUM(...)
💎	user_rating FLOAT
💎	release_date DATE
💎	image BLOB
Indexes ▶	

release_date	lang	prog_id	name	lang	release_date
2008-11-30	EN	20	Borgen	SE	2010-09-26
2015-10-10	EN	9	Wallander	EN	2008-11-30
2015-09-20	IS	25	The Last Kingdom	EN	2015-10-10
2010-09-26	SE	5	Ófærð	IS	2015-09-20

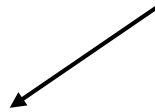
```
select * from tv_shows
where release_date > '2014-01-10' and lang like 'is'
```



Multi Column Index

- Column with higher cardinality should be placed first
 - Index cardinality means the number of unique values
 - A unique

release_date has a higher cardinality than lang
Should be placed at the left most



release_date	lang
2008-11-30	EN
2015-10-10	EN
2015-09-20	IS
2010-09-26	SE

prog_id	name	lang	release_date
20	Borgen	SE	2010-09-26
9	Wallander	EN	2008-11-30
25	The Last Kingdom	EN	2015-10-10
5	Ófærð	IS	2015-09-20



Using Multi Column Index

- Order is important

```
select * from tv_shows
  where release_date > '2014-01-01' and lang like 'is'
```

- Use of index from left to right

```
select * from tv_shows
  where release_date > '2010-01-01'
```



```
select * from tv_shows
  where lang = 'en'
```





Using Multi Column Index

user_rating

lang

rating

where **user_rating** > 5

where **user_rating** > 5 and **lang** = 'EN'

where **user_rating** > 5 and **lang** = 'EN' and **rating** = 'NC16'

where **lang** = 'EN'

where **lang** = 'EN' and **rating** = 'NC16'

where **lang** = 'EN' and **user_rating** > 5



Covering Index

- Results can be returned by just querying the index
 - Index contains all the columns to fulfil the query
 - The index holds all the data

```
create index lang_release_date_idx  
on tv_shows (release_date, lang);
```



```
select distinct(lang) from tv_shows  
where release_date > '2019-01-01'
```

Result can be
returned from query

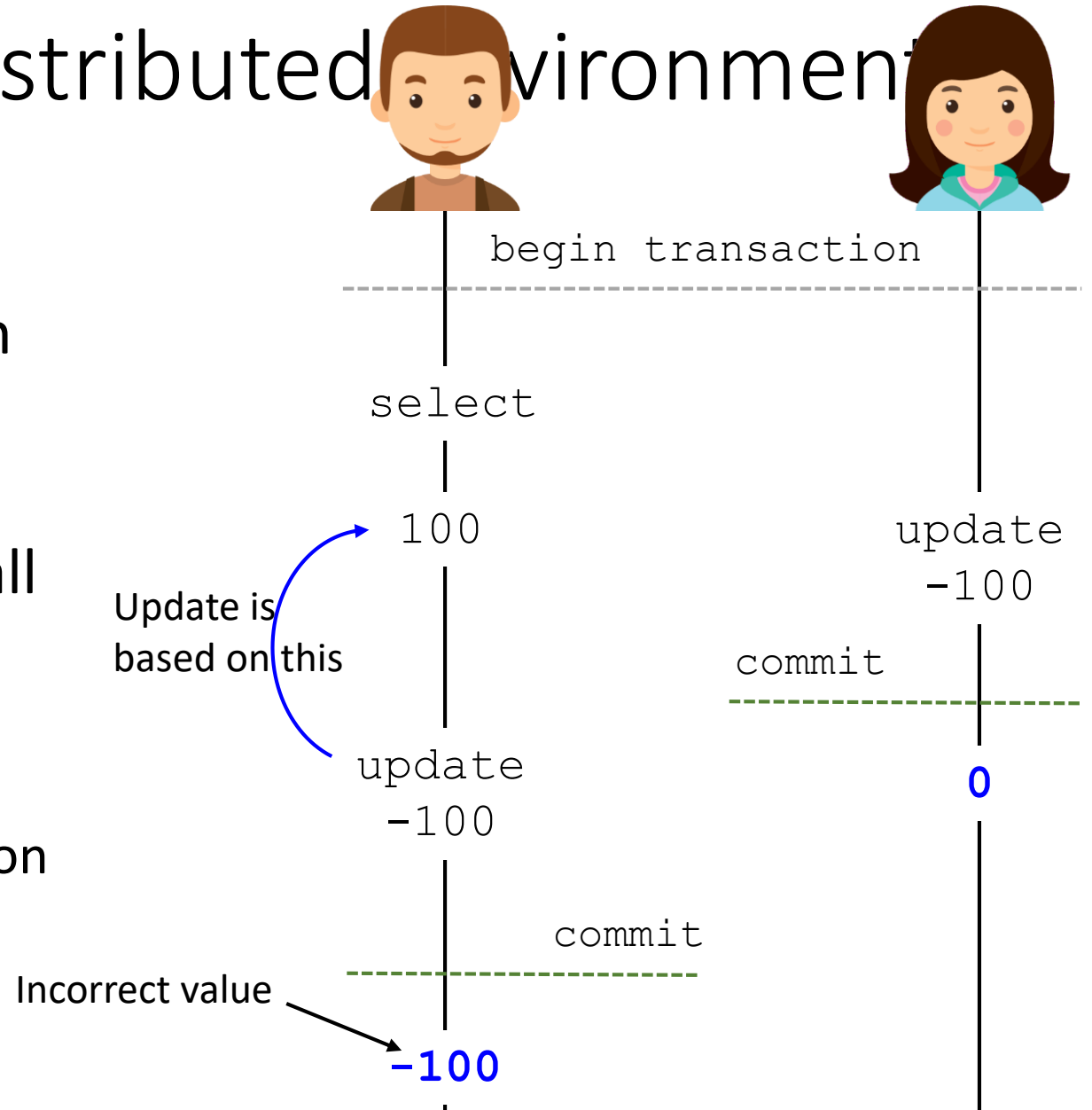


```
select release_date from tv_shows  
where lang like 'en'
```



Values Changes in Distributed Environment

- What happens when you perform a commit based on an old value?
 - When the record has changed
- Use pessimistic locks to eject all changes until you have completed your update
 - Bad because will slow the application because of contention





Conditional Update

- Updating a single record may be incorrect if the original value of the read has changed
- Perform the update based on an expected value of a field
 - Eg a timestamp
 - Compare and set method
- Eg. Withdrawing \$70 from an account
 - Current account balance is \$100
 - Update balance to \$30 provided the balance is still \$100

```
select @old_balance := balance,  
       @last_update := update_time  
from accounts  
where acct_id = 'abc123';
```

```
insert into accounts  
set balance = balance - @amount  
where balance = @old_balance and  
       update_time = @last_update
```



Implementing Conditional Update

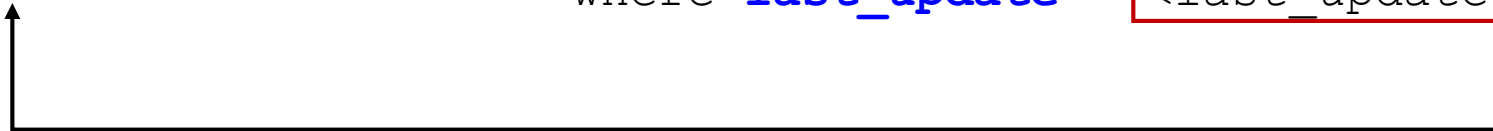
- Using a extra column to hold the timestamp of the last update
 - Column will be automatically updated when the record is updated

```
create table accounts (  
    acct_id varchar(8) not null,  
    balance decimal(10,2),  
    last_update default current_timestamp  
        on update current_timestamp,  
    primary key(acct_id)  
)
```

Update this field with the
timestamp whenever the record
is updated

```
select * from accounts  
where acct_id = 'abc123'
```

```
update accounts  
set balance = <new_balance>  
where last_update = <last_update from select>
```





Funds Transfer



Fred

Transfer \$100

Barney



Balance: \$500



Balance: \$300

 \$100



Balance: \$400

 \$100



Balance: \$400



Funds Transfer



Fred



Barney



Balance: \$500



Balance: \$300

 \$100



Balance: \$400



Inconsistent



Balance: \$300

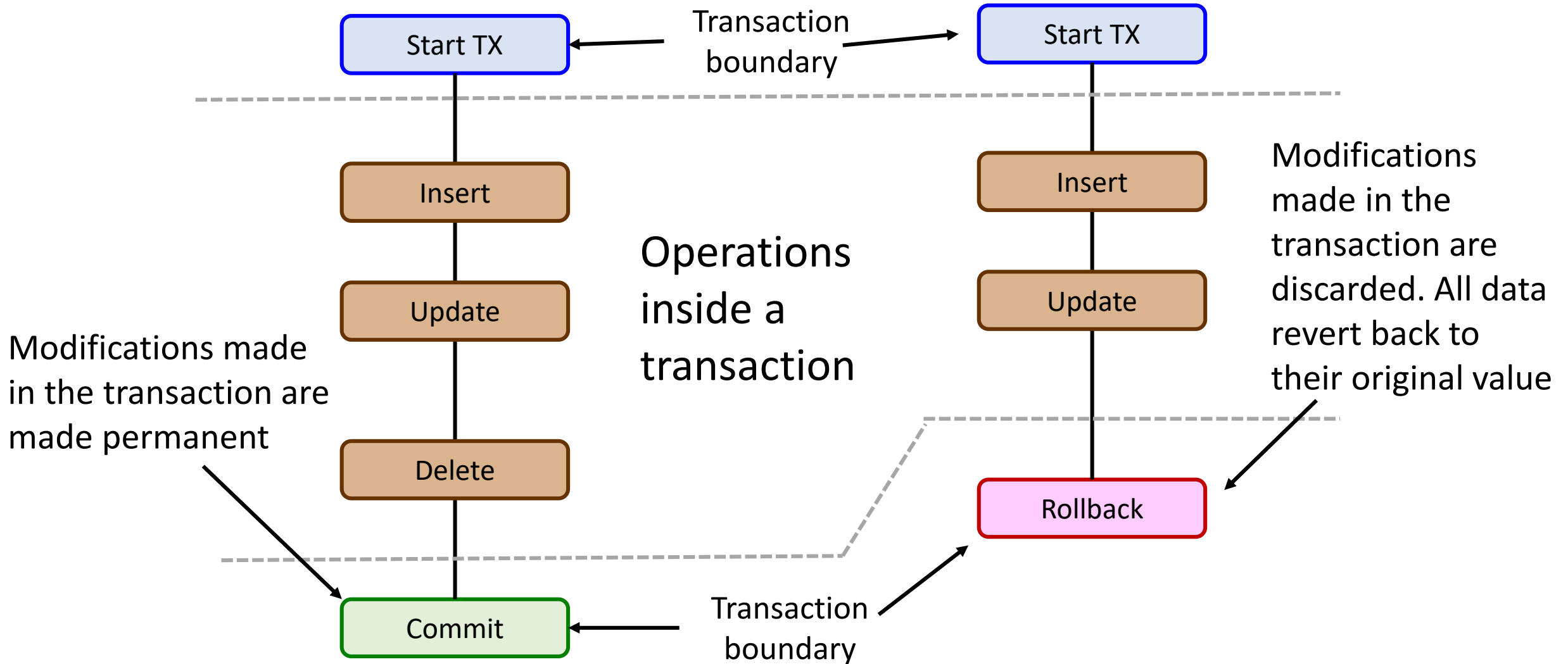


What are Transactions?

- Group multiple operations into a single atomic unit of work
 - Cannot be separated
- All operations within this group must succeed or not performed at all
 - In the event of anyone failing
 - Keep the database in a consistent state
- At the end of a transaction
 - Commit - save all the work performed inside the transaction
 - Rollback - discard all the work performed inside the transaction



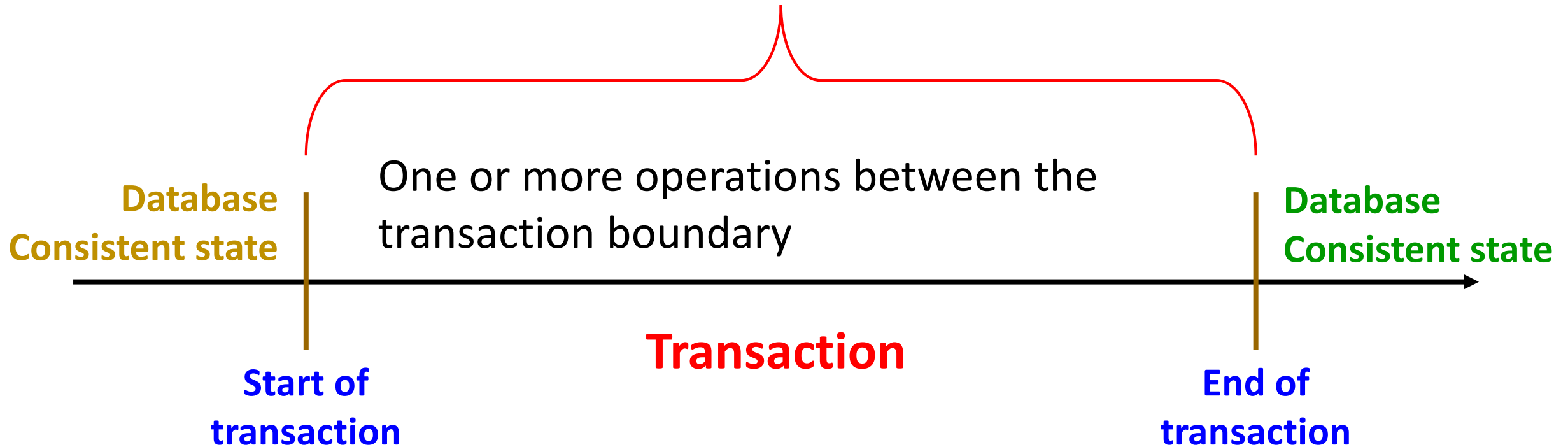
Transaction





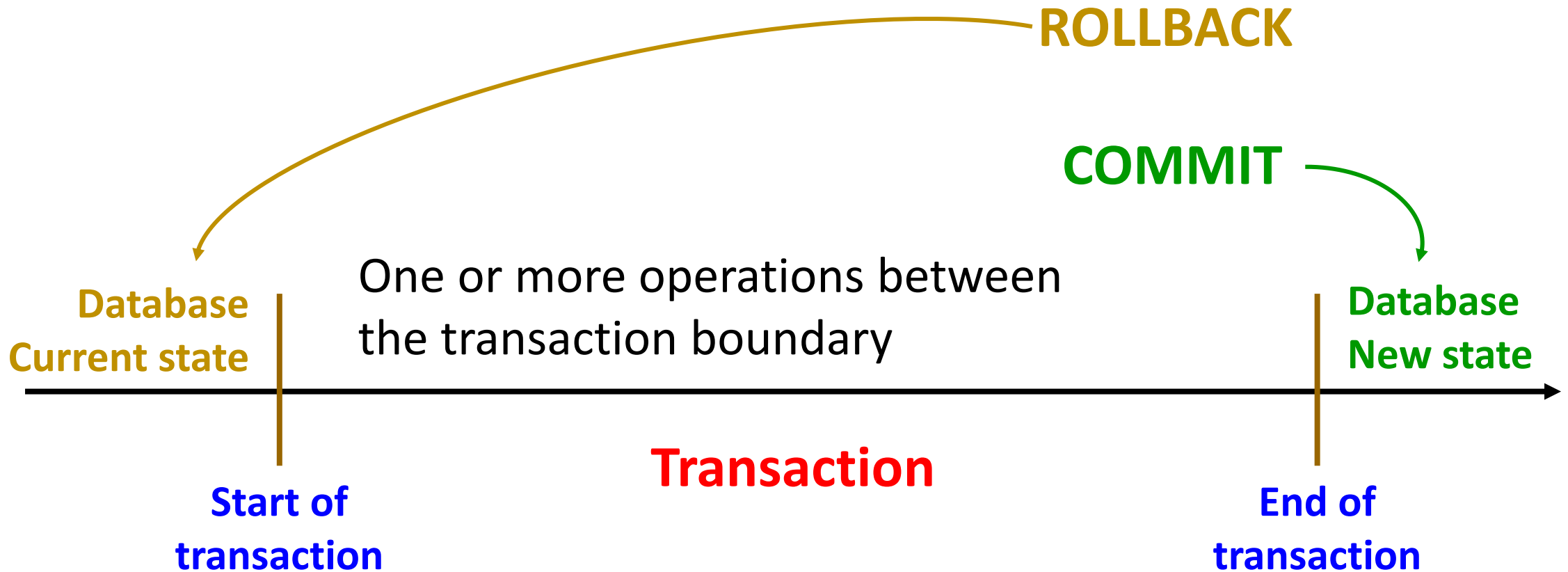
Transaction

All operations inside the transaction is considered atomic





Transaction





ACID Properties



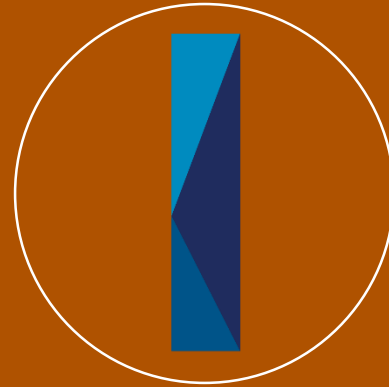
Atomicity

Each transaction is all or nothing



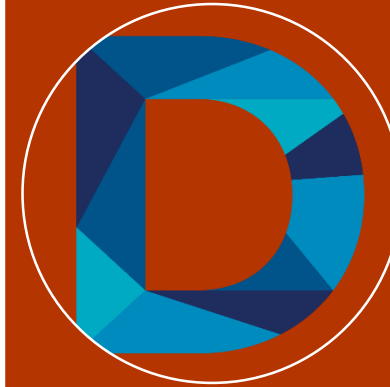
Consistency

Data should be valid according to the all defined rules



Isolation

Transactions do not affect each other



Durability

Committed data would not be lost



When to Use Transaction?

- When you have to modify related data from a few different tables from the same database
- If anyone of the operation fails, then must revert all the modified data back to its original value
 - Eg. Funds transfer from one account to another
- Prevent another operation from interfering with you operation before you commit your data
 - Eg. Prevent others from reading your new address before you finish the update



Transaction in SQL

```
set @from_acct = 'fred';  
set @to_acct = "barney";  
set @amount = 100;
```

Assume that there are enough funds
in `from_acct` for the transfer

pseudo code

```
start transaction
```

```
    update account set balance = balance - @amount  
        where acct_id = @from_acct;
```

```
    update account set balance = balance + @amount  
        where acct_id = @to_acct;
```

```
commit;
```




Performing Transaction

- Methods that are annotated with `@Transaction`
 - Typically methods in `@Service` but may also be present on other bean eg. `@Repository`
- Transaction are automatically committed when the method completes successfully
- Transaction will rollback when an unchecked exception is thrown
 - Exceptions subclass from `RuntimeException`
- Can specify a specific transaction with `rollbackFor` attribute



Example - Funds Transfer

begin transaction

update account set balance = balance - amount where acct_id = 'fred'

update account set balance = balance + amount where acct_id = 'barney'

commit



Example - Funds Transfer

```
@Repository
public class AccountRepository {

    @Autowired private JdbcTemplate template;

    // May throw unchecked exception DataAccessException
    public boolean withdraw(String acctId, double amount) {
        final int rowCount = template.update(
            "update account set balance = balance - ? where acct_id = ?",
            amount, accountId);
        return rowCount > 0;
    }

    public boolean deposit(String acctId, double amount) {
        final int rowCount = template.update(
            "update account set balance = balance + ? where acct_id = ?",
            amount, accountId);
        return rowCount > 0;
    }
}
```



Example - Funds Transfer

```
@Repository
public class AccountRepository {

    @Autowired private JdbcTemplate template;

    ...

    public Optional<double> getBalance(String acctId) {
        final SqlRowSet rs = template.query(
            'select balance from account where acct_id = ?', acctId);
        return Optional.ofNullable(
            rs.next()? rs.getDouble("balance"): null);
    }
}
```



Example - Funds Transfer

```
@Service
public class FundsTransfer {
    @Autowired AccountRepository acctRepo;
```



Transaction will rollback when these exceptions are thrown

@Transactional

```
public void transfer(String fromAcct, String toAcct, double amount) {

    final Optional<double> optFrom = acctRepo.getBalance(fromAcct);
    final Optional<double> optTo = acctRepo.getBalance(toAcct);
    if (optFrom.isEmpty() || optTo.isEmpty() || (optFrom.get() < amount))
        throw new IllegalArgumentException("Incorrect parameters");

    if !(acctRepo.withdraw(fromAcct, amount) ||
        acctRepo.deposit(toAcct, amount))
        throw new DataAccessException("Cannot perform transfer");

}
```

Transaction boundary

Transaction commits when exit.
Updates are now permanent



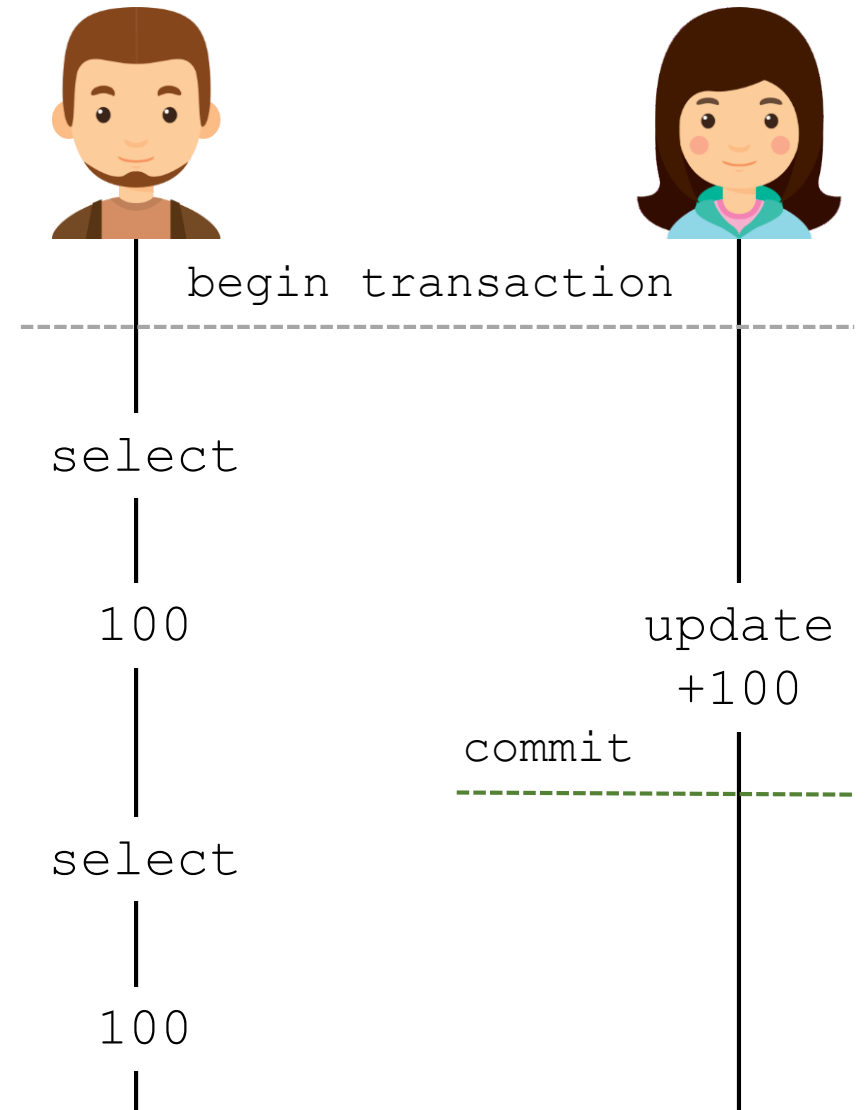


Unused



Isolation Level - Repeatable Read

- Refers to the visibility of uncommitted results inside a transaction
- Default is Repeatable Reads
- Reads of in flight transactions are isolated from changes
- Defaults to JDBC driver when not set
 - MySQL defaults to repeatable read (TRANSACTION_READ_COMMITTED)





Locks in Transaction

- MySQL will lock the required resources inside a transaction to ensure data consistency
- Row locking for tables with the required index(s)
 - Other sessions can access other rows except for the locked row

```
update account set balance = balance - amount  
where acct_id like 'fred'
```

Primary key has index

- Table locking if the table does not have the appropriate index(s)
 - No session can access any records from the table until transaction completes

```
update account set balance = balance - amount  
where name like 'fred'
```

Non primary key might
not have index