ACID

ACID jest skrótem od angielskich słów atomicity, consistency, isolation, durability, czyli niepodzielność, spójność, izolacja, trwałość.
Jest to zbiór właściwości gwarantujących poprawne przetwarzanie transakcji w bazach danych.

ATOMICITY

Niepodzielność transakcji oznacza jej wykonanie jako całości. Gdy transakcja powoduje wiele zmian w bazie danych możliwe są jedynie dwie sytuacje:
- wszystkie zmiany zostaną wykonane gdy transakcja zostanie zatwierdzona (instrukcja COMMIT)
- wszystkie zmiany zostaną wycofane (instrukcja ROLLBACK)

CONSISTENCY

Spójności bazy jest zachowana przez cały czas.
Zmiana stanu bazy może dokonać się jedynie za pomocą instrukcji COMMIT lub ROLLBACK. Jeśli w danej chwili jakaś transakcja zmienia dane w jednej
lub kilku tabelach inne działające w tym samym czasie transakcje zobaczą wszystkie nie zaktualizowane dane lub wszystkie nowe dane a nie zaś
mieszankę nowych i starych danych.

ISOLATION

Transakcje są zabezpieczone (odizolowane od siebie). Oznacza to, że podczas dziania nie mogą kolidować ze sobą lub zobaczyć nawzajem
niezatwierdzonych danych. Każda transakcja widzi bazę tak jakby była jedyną wykonywaną w danym czasie.
Izolacja jest osiągana za pomocą blokad. Zaawansowani użytkownicy bazy danych mogą dostosować poziom izolacji, zmniejszając je izolacji na rzecz zwiększenia wydajności i współbieżności.

DURABILITY

Rezultat wykonania transakcji musi być trwały. Jeśli operacja COMMIT powiodła się, jej rezultaty nie mogą ulec zmianie w przypadku awarii systemu baz danych.

READ UNCOMMITED

Poziom izolacji **READ UNCOMMITED** oznacza brak izolacji pomiędzy transakcjami. Transakcje mogą zobaczyć zmiany danych dokonane przez inne transakcje, które jeszcze nie zostały zatwierdzone. Oznacza to, że transakcje mogłyby odczytywać dane, które mogą ostatecznie nie istnieć w systemie, ponieważ inna transakcja, która aktualizowała dane, cofnęła zmiany i nie zatwierdziła ich. Systemu działającego w takim trybie nie można nazwać systemem transakcyjnym.

READ COMMITED

Poziom izolacji transakcji **READ COMMITED** określa że potwierdzone poleceniem COMMIT zmiany danych w tabelach są widoczne z poziomu innych transakcji. Oznacza to, że identyczne polecenia w obrębie tej samej transakcji mogą zwrócić zupełnie inne wyniki. W niektórych systemach baz danych jest to domyślny sposób izolacji transakcji.

REPEATABLE READ

Transakcja nie może czytać ani zapisywać na wierszach odczytywanych lub zapisywanych w innej transakcji. Wykonane polecenie SELECT zwraca wynik, który charakteryzuje się spójnością, a nowe rekordy dodane do tabeli z poziomu innej transakcji nie są od razu widoczne. Aby były widoczne należy bezwzględnie zakończyć transakcję.

SERIALIZABLE

Tryb **SERIALIZABLE** posuwa się o krok dalej niż tryb REPEATABLE READ. W trybie SERIALIZABLE wszystkie zwyczajne polecenia SELECT są traktowane jakby były wykonywane z klauzulą LOCK IN SHARE MODE.

Takie udostępnianie danych blokuje wszystkie zmiany danych (polecenia UPDATE i DELETE) i, jeśli ostatnie zmiany nie były jeszcze potwierdzone poleceniem COMMIT, powoduje oczekiwanie na wynik zapytania dopóty, dopóki nie nastąpi potwierdzenie transakcji w sesji, która rozpoczęła tą transakcję.

PROCEDURY

Prawie wszystkie procedury wymagają parametrów. Parametry sprawiają, że procedura składowana jest bardziej elastyczna i użyteczna. W MySQL parametr ma jeden z trzech trybów: IN, OUT lub INOUT.

PROCEDURY – PARAMETRY IN

IN - jest trybem domyślnym. Podczas definiowania parametru IN w procedurze składowanej program wywołujący musi przekazać argument do procedury składowanej. Ponadto wartość parametru IN jest chroniona. Oznacza to, że nawet jeśli wartość parametru IN jest zmieniana wewnątrz procedury, jego pierwotna wartość zostaje zachowana po zakończeniu procedury składowanej.

```
CREATE PROCEDURE `getCountryByName`(
        IN countryName VARCHAR(255))
BEGIN
        SELECT * FROM countries
        WHERE country_name = countryName;
END
```

PROCEDURY – PARAMETRY OUT

OUT - wartość parametru OUT może zostać zmieniona wewnątrz procedury składowanej, a jego nowa wartość jest przekazywana z powrotem do programu wywołującego.

```
CREATE PROCEDURE `getEmployeesCountByName`(IN employeeName
VARCHAR(255), OUT employeesCount INT)
BEGIN
   SELECT COUNT(employee_id)
   INTO employeesCount
   FROM employees
   WHERE first_name = employeeName;
END
```

PROCEDURY – PARAMETRY IN/OUT

Parametrem **INOUT** jest kombinacja parametrów IN i OUT. Oznacza to, że program wywołujący może przekazać argument, a procedura przechowywana może modyfikować parametr INOUT i przekazać nową wartość z powrotem do programu wywołującego.

```
CREATE PROCEDURE `set_counter`(INOUT count INT(4),IN inc INT(4))
BEGIN
 SET count = count + inc;
END
```

PROCEDURY - ZMIENNE

Zmienne w procedurach składowanych deklarujemy w następujący sposób:

DECLARE variable_name datatype(size) DEFAULT default_value;

Aby przypisać zmiennej wartość używamy następującej składni:

DECLARE total_count INT DEFAULT 0;
SET total_count = 10;

FUNKCJE

Funkcje różnią się od procedur tym że zwracają pojedynczą wartość i mogą być wywoływane nawet w pojedyńczych zapytaniach.

```
CREATE FUNCTION `function_name`(param1,param2,...)
RETURNS INTEGER
BEGIN
    RETURN 1;
END
```

FUNKCJE

```
CREATE FUNCTION `employeesCount`() RETURNS int(11)
BEGIN
    DECLARE total_count INT DEFAULT 0;

  SELECT COUNT(employee_id)
  INTO total_count
  FROM employees;

  RETURN total_count;
END
```

TRIGGERY (WYZWALACZE)

W MySQL wyzwalaczem jest zestaw instrukcji SQL, który jest wywoływany automatycznie, gdy następuje zmiana danych w skojarzonej tabeli. Uruchomienie może zostać wywołane przed lub po zmianie danych przez instrukcje INSERT, UPDATE lub DELETE. Przed wersją MySQL 5.7.2 można zdefiniować maksimum sześć wyzwalaczy dla każdej tabeli.

TRIGGERY (WYZWALACZE)

BEFORE INSERT - aktywowany przed wstawieniem danych do tabeli

AFTER INSERT - aktywowane po wstawieniu danych do tabeli

BEFORE UPDATE - aktywowany przed aktualizacją danych w tabeli

AFTER UPDATE - aktywowany po zaktualizowaniu danych w tabeli

BEFORE DELETE - aktywowany przed usunięciem danych z tabeli

AFTER DELETE - aktywowany po usunięciu danych z tabeli

TRIGGERY (WYZWALACZE)

```
CREATE TRIGGER trigger_name trigger_time trigger_event
 ON table_name
 FOR EACH ROW
 BEGIN
 END;
```

TRIGGERY (WYZWALACZE)

```
DROP TABLE IF EXISTS `hr`.`employees_audit`;

CREATE TABLE  `hr`.`employees_audit` (

 `id` int(10) unsigned NOT NULL AUTO_INCREMENT,

 `username` varchar(45) DEFAULT NULL,

 `operation` varchar(45) DEFAULT NULL,

 `employee_id` int(10) unsigned DEFAULT NULL,

 PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO_INCREMENT=0 DEFAULT CHARSET=utf8;
```

TRIGGERY (WYZWALACZE)

```
DELIMITER $$

CREATE TRIGGER before_employee_update
BEFORE UPDATE ON employees
FOR EACH ROW
BEGIN

   INSERT INTO employees_audit(username,operation,employee_id)
  VALUES(user(),'update', old.employee_id);

  DELETE FROM employees WHERE id = 10;

END$$
DELIMITER ;
```