Lebanese University
Faculty of Science-II
Department of Computer Science

I3306-Project DB

Fall 2023-2024

CodeQuest

Presented by: Nasr Ibrahim (56952)

Presented to: Dr. Hamssa Hasrouny

Contents

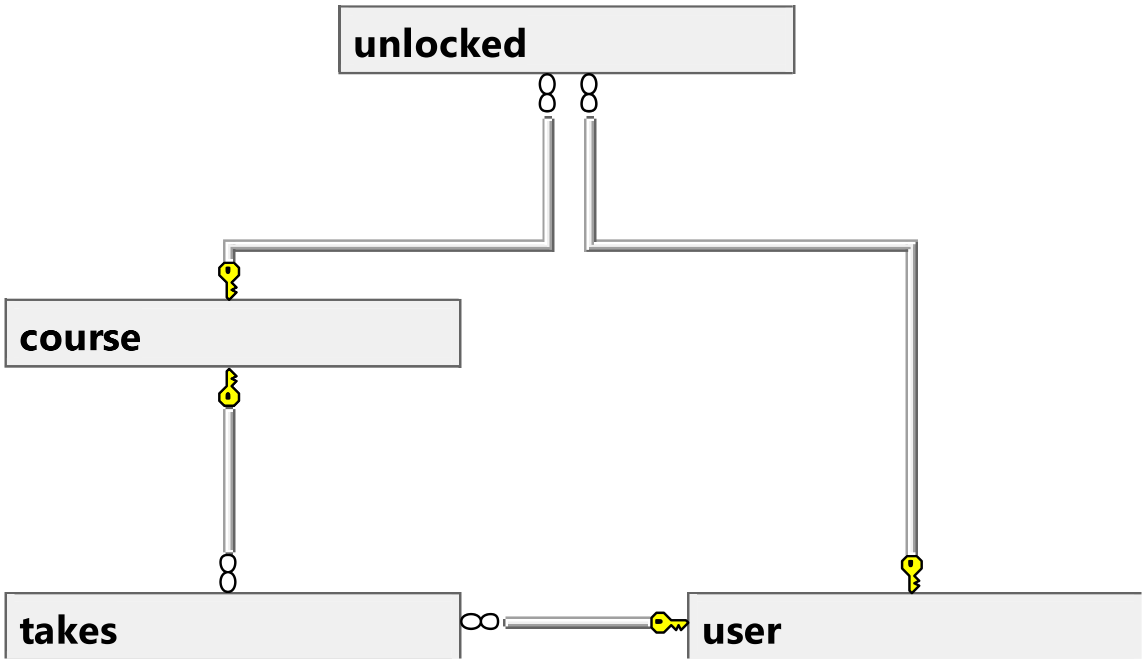
1. Introduction

In the rapidly evolving landscape of technology, the demand for specialized skills in areas such as web development, cybersecurity, and software engineering has surged. What started with a few individuals exploring the internet's potential has grown into entire teams of specialists addressing the evolving challenges. With technology becoming an integral part of daily life, even young children effortlessly interact with electronic devices. The implications of this early exposure on their future careers and thought processes are crucial, as the demand for tech-related jobs continues to rise. The journey to becoming a tech specialist doesn't begin in university; it starts when high school students consider these fields due to the opportunities they present.

Recognizing the need for a structured and effective approach to learning coding languages, and inspired by the simplicity of language learning app DuoLingo, CodeQuest aims to guide users through the complexities of coding languages by introducing them gradually. The app emphasizes building a strong foundation in one language before progressing to others. Users advance to the next language only after successfully mastering the current one through a series of tests. This sequential approach is designed to prevent burnout, instill a solid understanding of coding principles, and keep users motivated as they unlock new skills on their journey.

There are two aspects to the data structure and database storage: an online aspect and offline one. The offline is represented by a text file internal to the application itself and accessible directly through Java code. Here we will focus on the online database through MySQL. It mostly focuses on the relations between the users and lessons and arranges it to keep a close eye on the progress.

2. E/R Model



3. Relational Model

The entities are:

user (userId, username, password, name, phone, progress)
course (courseId, courseName)

There are two relations between them as follows:

takes (courseId, userId) represents the courses a user is currently taking. It is a many-to-many relation since multiple users can have multiple courses. In turn, takes has a one-to-many relation with each.

unlocked (courseId, userId, started) represents the courses a user has unlocked. It has the same relations as takes.

4. Queries

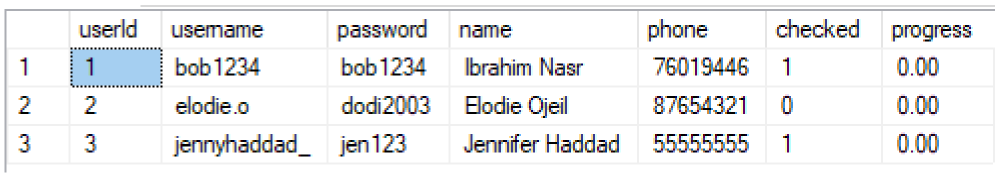
This section lists the queries used to create and alter the tables of the database

4.1 DDL Queries to create the tables

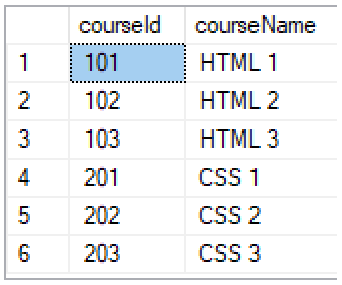
```
CREATE TABLE [dbo].user (
    userId INT NOT NULL,
    username VARCHAR(20) NOT NULL,
    password VARCHAR(20) NOT NULL,
    name VARCHAR(40) NULL,
    phone INT CHECK (phone >= 10000000 AND phone <= 99999999),
    checked BIT NULL,
    progress decimal(5, 2) NULL,
    CONSTRAINT PK_user PRIMARY KEY CLUSTERED (userId ASC),
    CONSTRAINT DF_user_name DEFAULT NULL FOR name,
    CONSTRAINT DF_user_phone DEFAULT NULL FOR phone,
    CONSTRAINT DF_user_checked DEFAULT (0) FOR checked,
    CONSTRAINT DF_user_progress DEFAULT (0) FOR progress
);

CREATE TABLE [dbo].course (
    courseId INT NOT NULL,
    courseName VARCHAR(10) NOT NULL,
    PRIMARY KEY (courseId)
);

CREATE TABLE [dbo].[takes](
    [courseId] [int] NOT NULL,
    [userId] [int] NOT NULL,
    CONSTRAINT [FK_takes_course1] FOREIGN KEY([courseId]) REFERENCES [dbo].[course]
([courseId]),
    CONSTRAINT [FK_takes_user1] FOREIGN KEY([userId]) REFERENCES [dbo].[user] ([userId])
);

CREATE TABLE [dbo].unlocked (
    courseId INT NOT NULL,
    userId INT NOT NULL,
    started BIT NULL,
    CONSTRAINT FK_unlocked_course FOREIGN KEY (courseId) REFERENCES course (courseId),
    CONSTRAINT FK_unlocked_user FOREIGN KEY (userId) REFERENCES user (userId)
);
```

4.2 Inserted Values

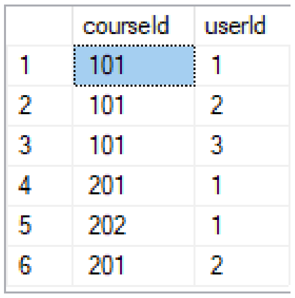
The following values were initially inserted into the tables:



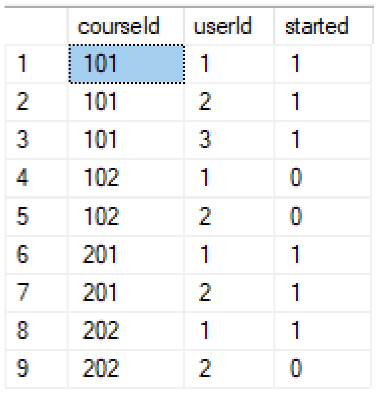
	userId	username	password	name	phone	checked	progress
1	1	bob1234	bob1234	Ibrahim Nasr	76019446	1	0.00
2	2	elodie.o	dodi2003	Elodie Ojeil	87654321	0	0.00
3	3	jennyhaddad_	jen123	Jennifer Haddad	55555555	1	0.00



	courseId	courseName
1	101	HTML 1
2	102	HTML 2
3	103	HTML 3
4	201	CSS 1
5	202	CSS 2
6	203	CSS 3



	courseId	userId
1	101	1
2	101	2
3	101	3
4	201	1
5	202	1
6	201	2



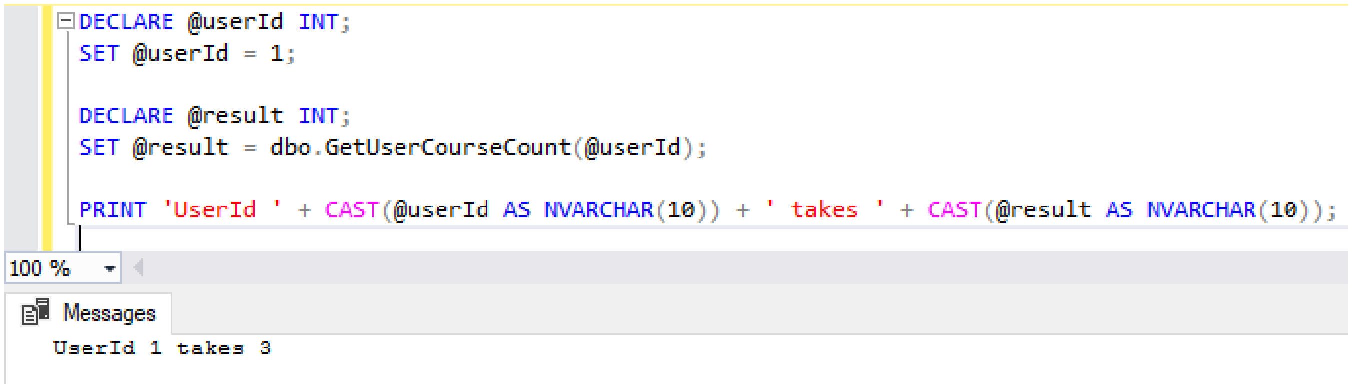
	courseId	userId	started
1	101	1	1
2	101	2	1
3	101	3	1
4	102	1	0
5	102	2	0
6	201	1	1
7	201	2	1
8	202	1	1
9	202	2	0

4.3 Function Queries

4.3.1 To get course count of specific user

```
CREATE FUNCTION dbo.GetUserCourseCount(@userId INT)
RETURNS INT
AS
BEGIN
    DECLARE @courseCount INT;

    SELECT @courseCount = COUNT(*)
    FROM takes
    WHERE userId = @userId;

    RETURN @courseCount;
END;
```

The following is the call and printed result



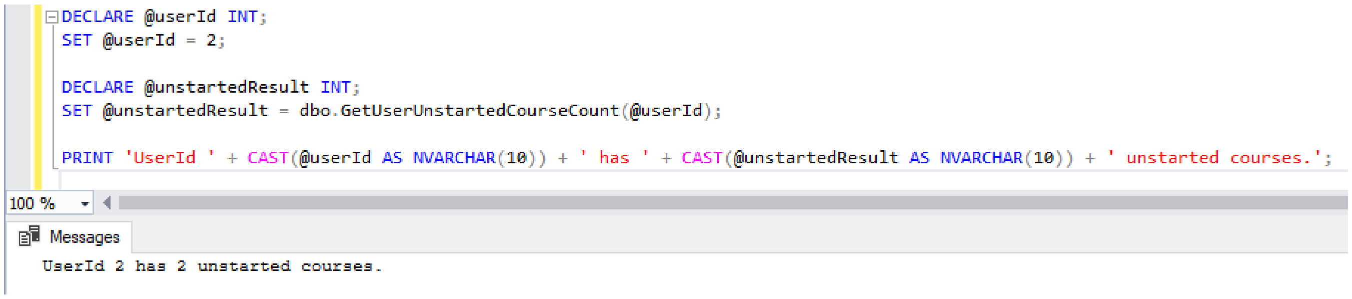
```
DECLARE @userId INT;
SET @userId = 1;

DECLARE @result INT;
SET @result = dbo.GetUserCourseCount(@userId);

PRINT 'UserId ' + CAST(@userId AS NVARCHAR(10)) + ' takes ' + CAST(@result AS NVARCHAR(10));
```

```
UserId 1 takes 3
```

4.3.2 Get unlocked courses not started

```
CREATE FUNCTION dbo.GetUserUnstartedCourseCount(@userId INT)
RETURNS INT
AS
BEGIN
    DECLARE @unstartedCount INT;

    SELECT @unstartedCount = COUNT(*)
    FROM unlocked
    WHERE userId = @userId AND started = 0;

    RETURN @unstartedCount;
END;
```

The following is the call and printed result



```
DECLARE @userId INT;
SET @userId = 2;

DECLARE @unstartedResult INT;
SET @unstartedResult = dbo.GetUserUnstartedCourseCount(@userId);

PRINT 'UserId ' + CAST(@userId AS NVARCHAR(10)) + ' has ' + CAST(@unstartedResult AS NVARCHAR(10)) + ' unstarted courses.';
```

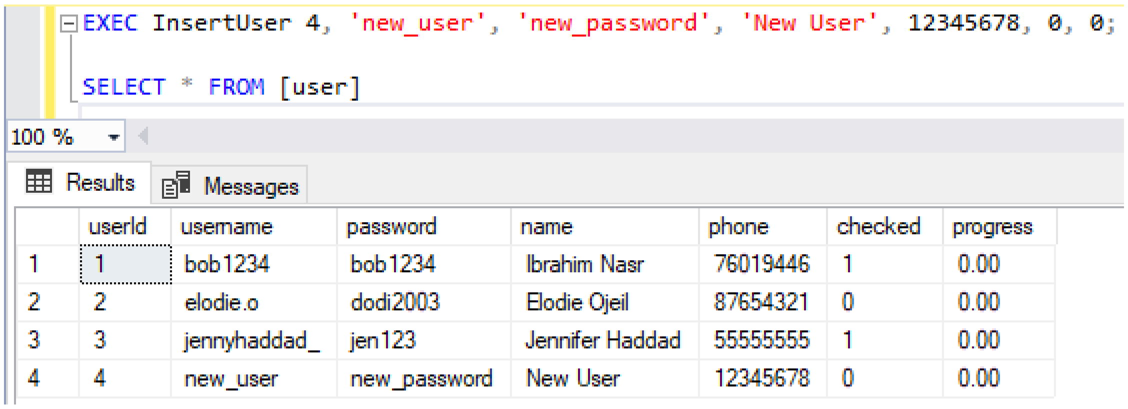
```
UserId 2 has 2 unstarted courses.
```

4.4 Procedure Queries

4.4.1 Add new user

```
CREATE PROCEDURE InsertUser
    @userId INT,
    @username VARCHAR(20),
    @password VARCHAR(20),
    @name VARCHAR(40),
    @phone NCHAR(10),
    @checked BIT,
    @progress DECIMAL(5,2)
AS
BEGIN
    INSERT INTO [user] (userId, username, password, name, phone, checked, progress)
    VALUES (@userId, @username, @password, @name, @phone, @checked, @progress);
END;
```

The following is the execution and result:



```
EXEC InsertUser 4, 'new_user', 'new_password', 'New User', 12345678, 0, 0;

SELECT * FROM [user]
```

100 %

Results | Messages

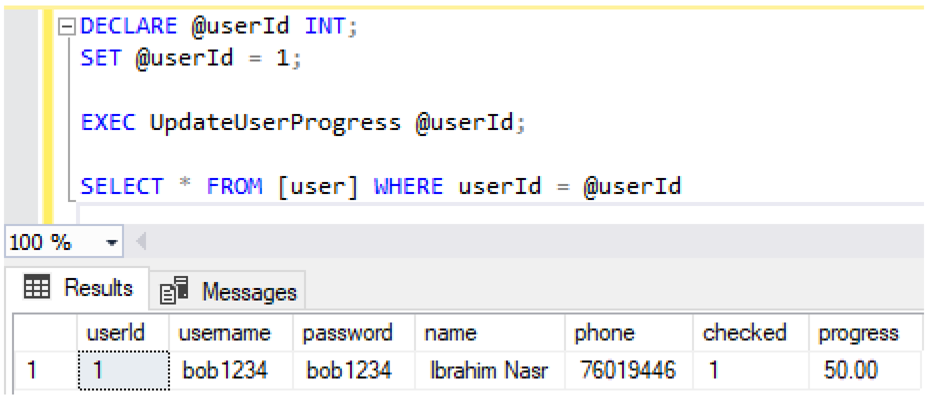
	userId	username	password	name	phone	checked	progress
1	1	bob1234	bob1234	Ibrahim Nasr	76019446	1	0.00
2	2	elodie.o	dodi2003	Elodie Ojeil	87654321	0	0.00
3	3	jennyhaddad_	jen123	Jennifer Haddad	55555555	1	0.00
4	4	new_user	new_password	New User	12345678	0	0.00

4.4.2 Update progress

```
CREATE PROCEDURE UpdateUserProgress
    @userId INT
AS
BEGIN
    DECLARE @unstartedCount INT;
    DECLARE @totalCourses INT;
    DECLARE @progress DECIMAL(5, 2);

    -- Call the function to get the count of started courses
    SET @unstartedCount = dbo.GetUserCourseCount(@userId);

    -- Get the total number of courses
    SELECT @totalCourses = COUNT(*)
    FROM course;

    -- Calculate the progress
    SET @progress = CASE
        WHEN @totalCourses > 0 THEN CONVERT(DECIMAL(5, 2), @unstartedCount) / @totalCourses *
100
        ELSE 0
    END;

    -- Update the progress in the user table
    UPDATE [user]
    SET progress = @progress
    WHERE userId = @userId;
```

```
END;
```

The following is the execution and result:



```
DECLARE @userId INT;
SET @userId = 1;

EXEC UpdateUserProgress @userId;

SELECT * FROM [user] WHERE userId = @userId
```

	userId	username	password	name	phone	checked	progress
1	1	bob1234	bob1234	Ibrahim Nasr	76019446	1	50.00

4.5 Trigger Queries

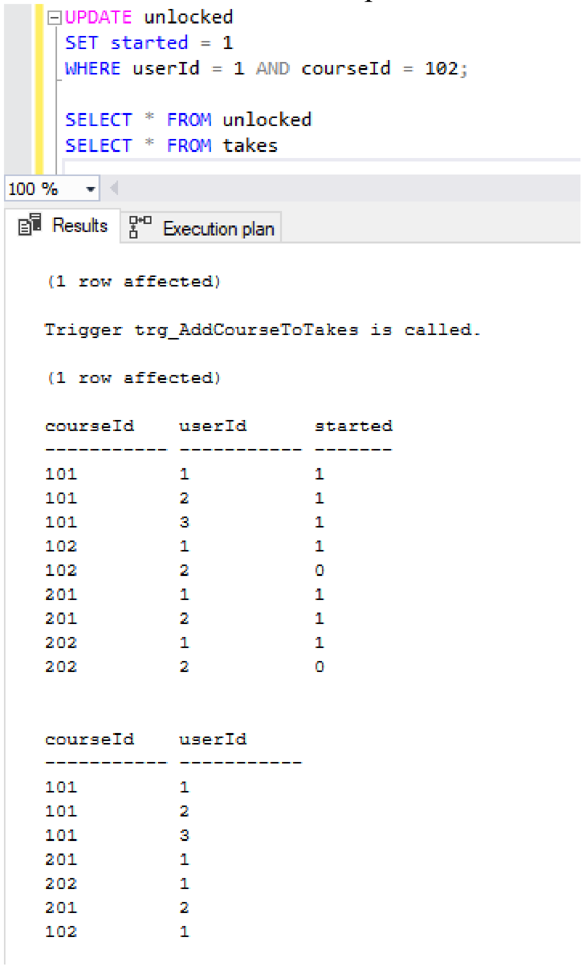
4.5.1 Add course to “takes”

This trigger applies to table “unlocked” where the rows that have column “started” updated to 1 are directly added to table “takes”.

```
CREATE TRIGGER trg_AddCourseToTakes
ON unlocked
AFTER UPDATE
AS
BEGIN
    IF UPDATE(started)
    BEGIN
        INSERT INTO takes (courseId, userId)
        SELECT i.courseId, i.userId
        FROM inserted i
        WHERE i.started = 1;
    END
END;
```

Notice that row 4, with courseId = 102 and userId = 1 now has started = 1 after being previously 0.

And the trigger is shown to have been called (through a debugging print message later added) and the courseId and userId are now automatically added in row 7 of “takes”.



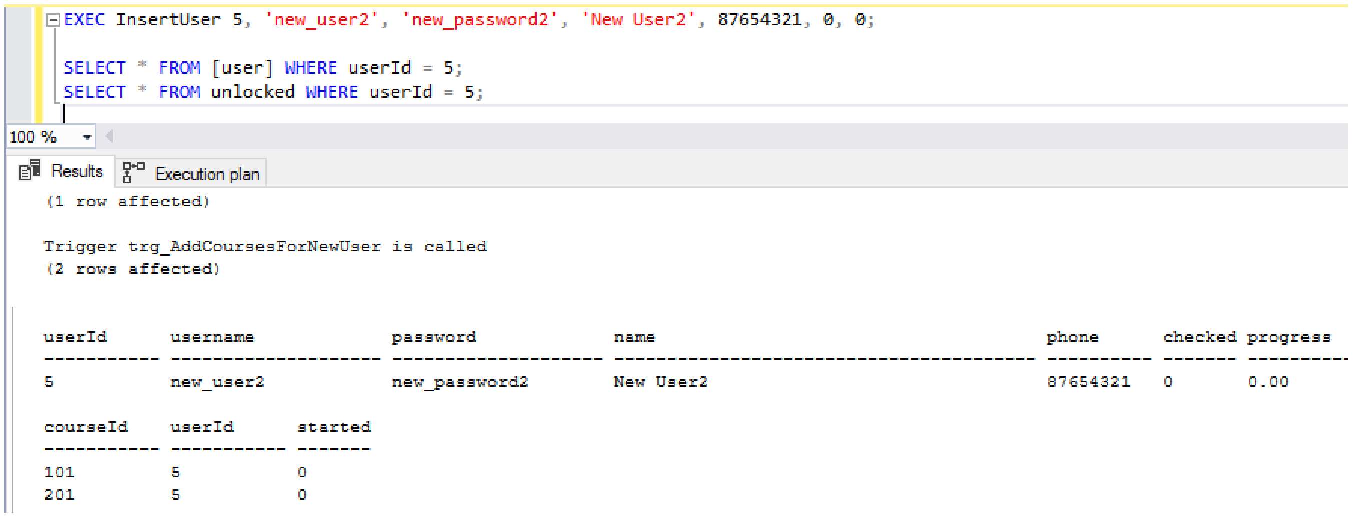
```
UPDATE unlocked
SET started = 1
WHERE userId = 1 AND courseId = 102;

SELECT * FROM unlocked
SELECT * FROM takes
```

```
(1 row affected)

Trigger trg_AddCourseToTakes is called.

(1 row affected)

courseId    userId      started
----------- ----------- -------
101         1           1
101         2           1
101         3           1
102         1           1
102         2           0
201         1           1
201         2           1
202         1           1
202         2           0


courseId    userId
----------- -----------
101         1
101         2
101         3
201         1
202         1
201         2
102         1
```

4.5.2 Add initial courses automatically to new users

```
CREATE TRIGGER trg_AddCoursesForNewUser
ON [user]
AFTER INSERT
AS
BEGIN
    INSERT INTO unlocked (courseId, userId, started)
    SELECT courseId, userId, 0
    FROM inserted
    CROSS JOIN (
        VALUES
            (101),
            (201)
    ) AS specific_courses(courseId);
END;
```

As shown below, when the procedure was executed, a new user was inserted. Automatically the trigger was called and the courses were added, facilitating the initialization process for new users.



```
EXEC InsertUser 5, 'new_user2', 'new_password2', 'New User2', 87654321, 0, 0;

SELECT * FROM [user] WHERE userId = 5;
SELECT * FROM unlocked WHERE userId = 5;
```

```
(1 row affected)

Trigger trg_AddCoursesForNewUser is called
(2 rows affected)


userId      username             password             name                                     phone      checked progress
----------- -------------------- -------------------- ---------------------------------------- ---------- ------- ----------
5           new_user2            new_password2        New User2                                87654321   0       0.00

courseId    userId      started
----------- ----------- -------
101         5           0
201         5           0
```

5. Conclusion

This database clarifies the entities and their main information facilitating their identification. In addition, the relations used clearly convey the purpose of the application and would facilitate the logical flow.

Moreover, as shown, the functions, procedures, and triggers, in the way they are interconnected would really facilitate the follow up of the database and data verification. In addition, this would help simplify the Java code in the application-side and the PHP code in the server-side.