LUT University

LAND OF THE CURIOUS

CT60A4304 - BASICS OF DATABASE SYSTEMS

SQL, TABLE MANAGEMENT

Lecture

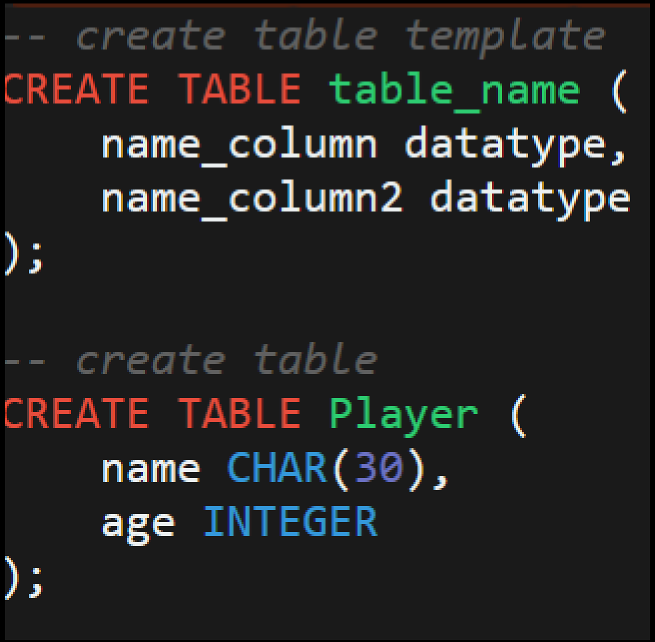
Jiri Musto, D.Sc.

TABLE OF CONTENTS

- Managing SQL tables
  - Create, alter, drop
- SQL Data types
  - String, numeric, datetime
- Constraints
  - Column constraints, reference constraints
- Automatic incrementation
- Triggers
- Views

CREATE

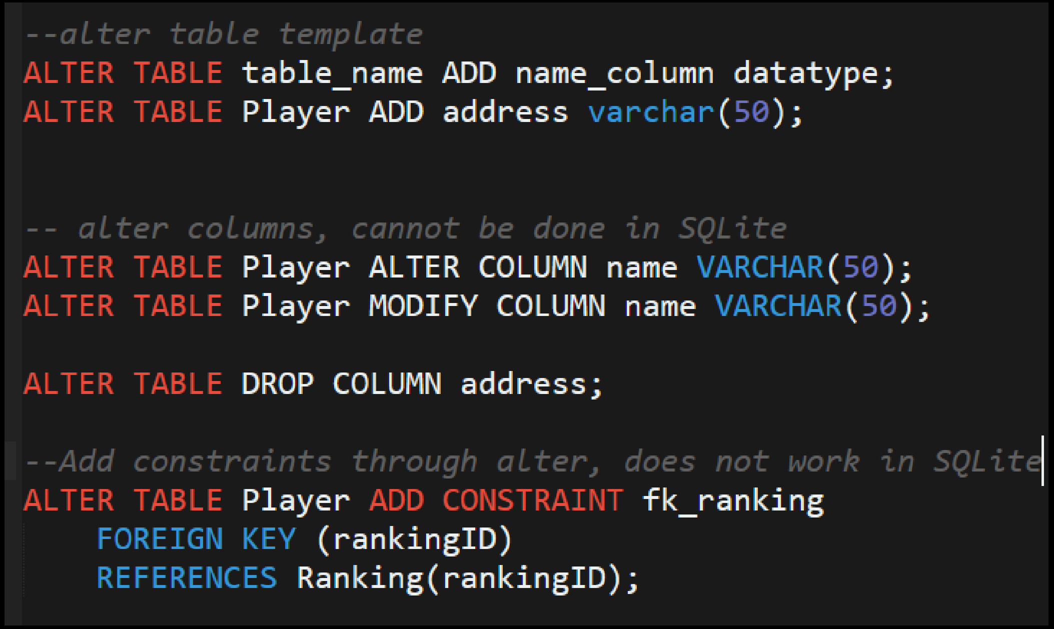
- Create a new table (or database)
- Can create tables based on other tables
- Cannot use reserved keywords for table or column names



```sql
-- create table template
CREATE TABLE table_name (
    name_column datatype,
    name_column2 datatype
);

-- create table
CREATE TABLE Player (
    name CHAR(30),
    age INTEGER
);
```

ALTER

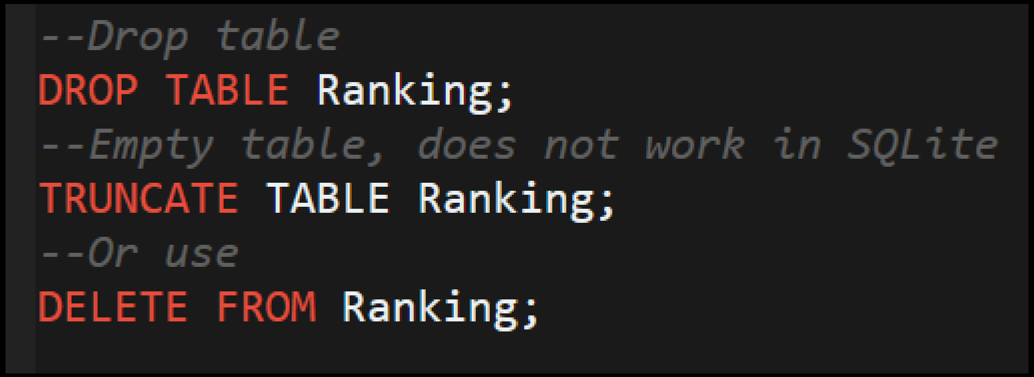
- Edit an existing table
  - Add new columns, constraints
  - Edit existing columns (names, types), remove columns



```sql
--alter table template
ALTER TABLE table_name ADD name_column datatype;
ALTER TABLE Player ADD address varchar(50);


-- alter columns, cannot be done in SQLite
ALTER TABLE Player ALTER COLUMN name VARCHAR(50);
ALTER TABLE Player MODIFY COLUMN name VARCHAR(50);

ALTER TABLE DROP COLUMN address;

--Add constraints through alter, does not work in SQLite
ALTER TABLE Player ADD CONSTRAINT fk_ranking
    FOREIGN KEY (rankingID)
    REFERENCES Ranking(rankingID);
```

DROP

- Drop table (or database)
- Can use another command to empty the table



```
--Drop table
DROP TABLE Ranking;
--Empty table, does not work in SQLite
TRUNCATE TABLE Ranking;
--Or use
DELETE FROM Ranking;
```

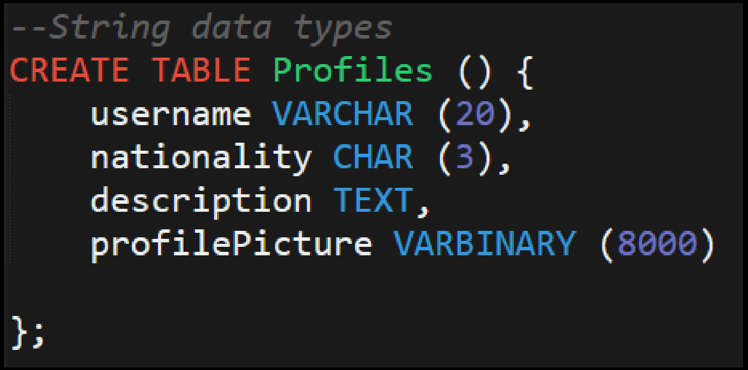
TABLE VS. ROW COMMANDS

	Table	Row
Create a new data.	CREATE	INSERT
Modify existing data	ALTER	UPDATE
Remove data	DROP	DELETE

LUT University

DATA TYPES: STRING

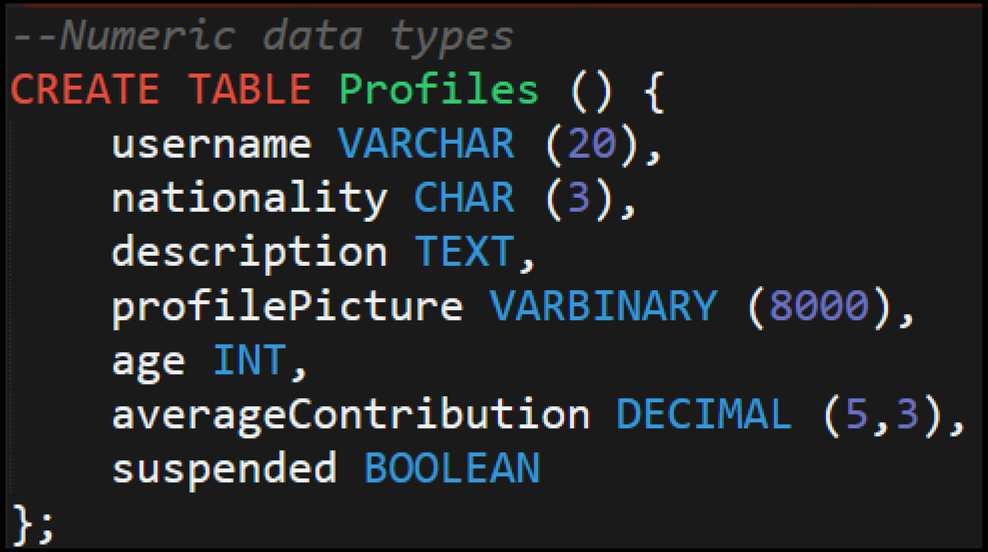
- CHAR(size) / VARCHAR (size), most common data type
  - Char is max 255 characters, Varchar is 65 535
- BINARY(size) / VARBINARY(size)
  - Same as Char/Varchar, stores binary byte strings
- TEXT(size)
- Array types: Have one or multiple values stored (chosen from a list)
  - SET and ENUM in MySQL
  - Collection in PostgreSQL, ENUM types
  - ARRAY in DB2
  - MULTISET in SQL Server



```
--String data types
CREATE TABLE Profiles () {
    username VARCHAR (20),
    nationality CHAR (3),
    description TEXT,
    profilePicture VARBINARY (8000)

};
```

DATA TYPES: NUMERIC

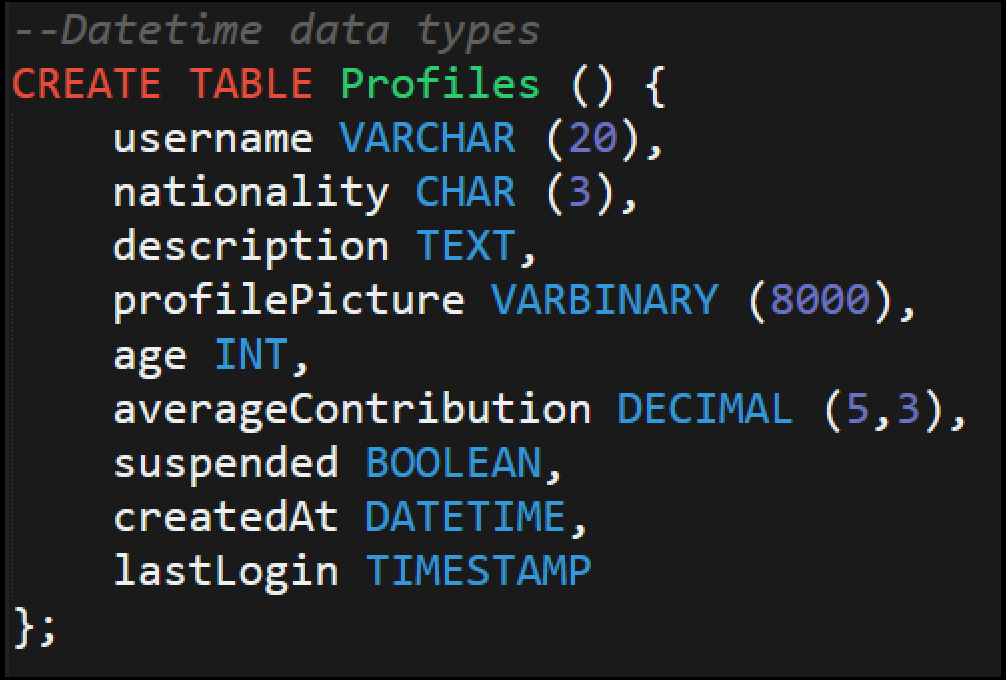
- INTEGER, most common data type
  - Between -2B to +2B
  - Most DBMS include smallint and bigint
- DECIMAL (size, digits) / NUMERIC
- BOOLEAN
  - True / false
  - Exists in some DBMS
  - Can use BIT type to imitate



```
--Numeric data types
CREATE TABLE Profiles () {
    username VARCHAR (20),
    nationality CHAR (3),
    description TEXT,
    profilePicture VARBINARY (8000),
    age INT,
    averageContribution DECIMAL (5,3),
    suspended BOOLEAN
};
```

DATA TYPES: DATETIME

- DATE
  - YYYY-MM-DD
- DATETIME:
  - YYYY-MM-DD hh:mm:ss
- TIME
  - hh:mm:ss
- TIMESTAMP
  - Possible to save the time when data is stored using system time
- DMBS offer different ways to format the date
  - it is possible to show date in ‘dd/mm/yyyy’ format



```
--Datetime data types
CREATE TABLE Profiles () {
    username VARCHAR (20),
    nationality CHAR (3),
    description TEXT,
    profilePicture VARBINARY (8000),
    age INT,
    averageContribution DECIMAL (5,3),
    suspended BOOLEAN,
    createdAt DATETIME,
    lastLogin TIMESTAMP
};
```

DATA TYPES: ADDITIONAL INFORMATION

- Many DBMS have their own unique data types
  - Such as: nchar, nvarchar, tinytext, smallint, image, smallmoney, xml, etc.
  - Be sure to check their documentation!
- SQLite (unlike other SQL DBMS) offers a dynamic data type
  - Determined by the value rather than the column
  - The type defined in SQLite is ‘recommended’, not ‘required’
  - SQLite uses the following dynamic data types:
    - Integer, text, blob, real, and numeric
    - Converts specific data types into more generic form
- It is better to get used to using specific types for SQL rather than the SQLite approach, unless you decide to only use SQLite
- Here is a good comparison of SQL Server, MySQL, PostgreSQL, Oracle data type: https://www.w3resource.com/sql/data-type.php

COLUMN CONSTRAINTS

- There are multiple integrity constraints in SQL to ensure data is consistent across the database
- PRIMARY KEY / FOREIGN KEY
- (NOT) NULL
- UNIQUE
- CHECK
- DEFAULT
- INDEX

PRIMARY KEY

- Select one (or multiple attributes) that form the primary key for the table
  - Multiple attributes form a ‘composite key’ as the primary key
- Table can have only one primary key

```
--PK
CREATE TABLE Profiles () {
    username VARCHAR (20) NOT NULL PRIMARY KEY,
    nationality CHAR (3),
    description TEXT,
    profilePicture VARBINARY (8000),
    age INT,
    averageContribution DECIMAL (5,3),
    suspended BOOLEAN,
    createdAt DATETIME,
    lastLogin TIMESTAMP
};
```

Option 1

```
CREATE TABLE Profiles () {
    username VARCHAR (20) NOT NULL,
    nationality CHAR (3),
    description TEXT,
    profilePicture VARBINARY (8000),
    age INT,
    averageContribution DECIMAL (5,3),
    suspended BOOLEAN,
    createdAt DATETIME,
    lastLogin TIMESTAMP,
    PRIMARY KEY (username)
};
```

Option 2

FOREIGN KEY

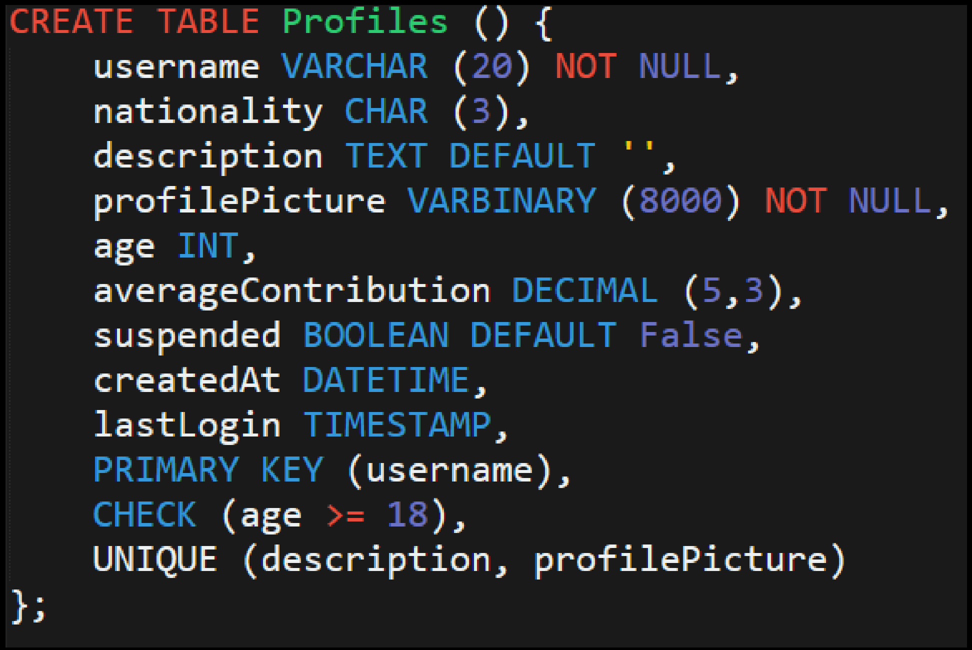
- Foreign key references another table and its primary key
- Restricts value set to what the referenced table has
  - I.e. Has to be the same data type
- The referenced table / row is called ‘parent’ while the referencing table/row is ‘child’

```
CREATE TABLE FriendList () {
    profile VARCHAR (20) NOT NULL,
    friend VARCHAR (20) NOT NULL,
    CONSTRAINT friendList_PK PRIMARY KEY (profile, fiend),
    FOREIGN KEY (profile) REFERENCES Profiles (username),
    FOREIGN KEY (friend) REFERENCES Profiles (username)
}
```

LUT University

COLUMN CONSTRAINTS

- When creating tables, the column values can be restricted using the following:
- CHECK: Restrict possible values using SQL comparison
- DEFAULT: Give a default value to a column that is automatically given if no other value is specified
- NOT NULL: Do not allow null or empty values.
- UNIQUE: No duplicate values are allowed in the table. Can be restricted to multiple columns
- INDEX: Covered in another lecture.



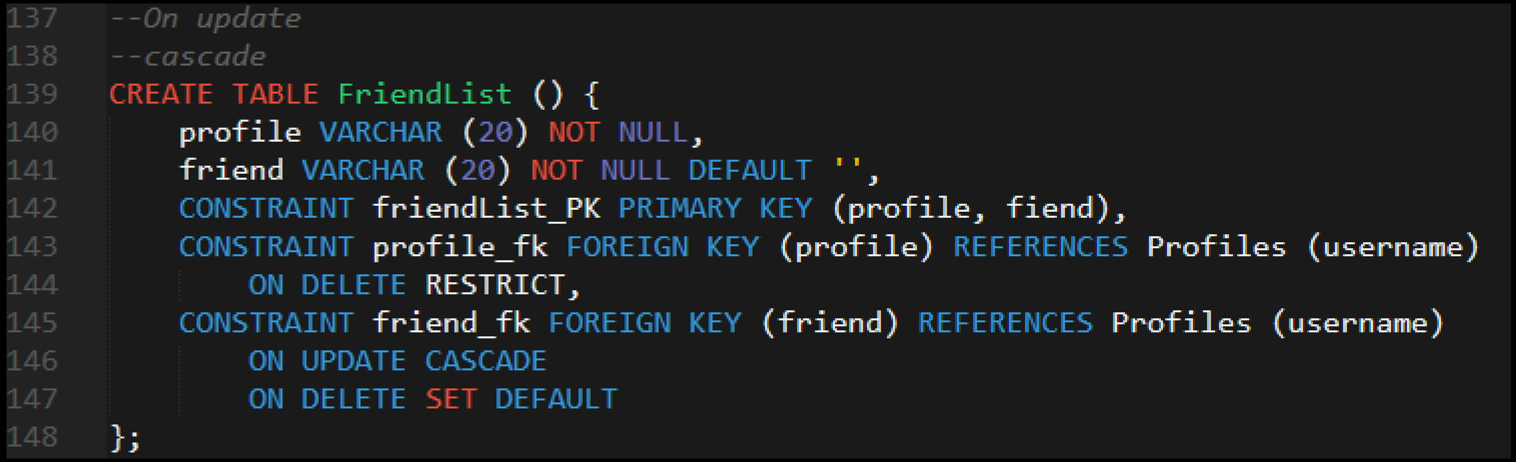
```
CREATE TABLE Profiles () {
    username VARCHAR (20) NOT NULL,
    nationality CHAR (3),
    description TEXT DEFAULT '',
    profilePicture VARBINARY (8000) NOT NULL,
    age INT,
    averageContribution DECIMAL (5,3),
    suspended BOOLEAN DEFAULT False,
    createdAt DATETIME,
    lastLogin TIMESTAMP,
    PRIMARY KEY (username),
    CHECK (age >= 18),
    UNIQUE (description, profilePicture)
};
```

REFERENCE CONSTRAINTS: ON UPDATE / DELETE

- Integrity rules are possible to be enforced using reference key (foreign key). The two triggers can be specified:
  - ON UPDATE: When the referenced value is updated
  - ON DELETE: When the referenced value is deleted
- There are five possible action that can be tied to the integrity rules:
  - NO ACTION: No action is taken. May raise an error and abort upon deletion.
  - RESTRICT: The ‘parent’ cannot by modified if any ‘children’ exist
  - CASCADE: On update, sets the existing value to the new one. On delete, removes the whole row.
  - SET NULL: Sets the referenced value to NULL. Requires the value to be nullable.
  - SET DEFAULT: Sets the referenced value to the given default value.

ON UPDATE / DELETE EXAMPLE

- When creating a constraint, use keyword CONSTRAINT
  - Usually not used when creating simple PK or FK
  - Used when adding ON DELETE/UPDATE constraints (may work without it but better be safe)



```sql
--On update
--cascade
CREATE TABLE FriendList () {
    profile VARCHAR (20) NOT NULL,
    friend VARCHAR (20) NOT NULL DEFAULT '',
    CONSTRAINT friendList_PK PRIMARY KEY (profile, fiend),
    CONSTRAINT profile_fk FOREIGN KEY (profile) REFERENCES Profiles (username)
        ON DELETE RESTRICT,
    CONSTRAINT friend_fk FOREIGN KEY (friend) REFERENCES Profiles (username)
        ON UPDATE CASCADE
        ON DELETE SET DEFAULT
};
```

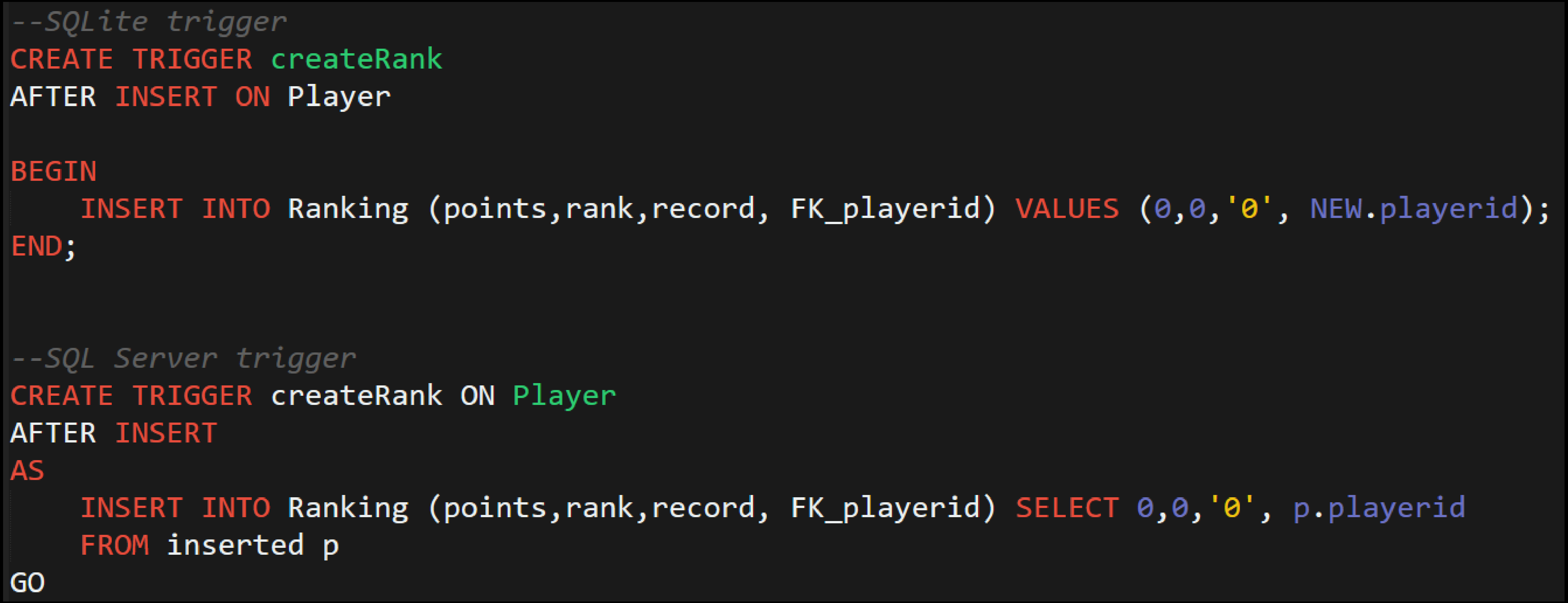
AUTOMATICALLY INCREMENTING ID

- Many relational DBMS offer an automatically incrementing ID
  - SQLite: ROWID and AUTOINCREMENT
  - MySQL: AUTO_INCREMENT
  - SQL Server: IDENTITY
  - PostgreSQL: SERIAL
- Allows the user to not worry about setting a unique identifier
  - Can set an ID column to automatically increment
  - Useful if data has no unique identifier already
  - May not always allow the user to specify the ID
  - Restricts the ID to a number

TRIGGER

- Triggers are SQL queries that automatically execute when conditions are met
  - BEFORE / AFTER / INSTEAD OF INSERT / UPDATE / DELETE
  - Similar to ON DELETE or ON UPDATE
- Are programmable, essentially functions of SQL
- Can be used for example:
  - Update multiple tables with one statement
  - Restrict columns to specific values
- DBMS dependent ways to refer to the existing data and new data
- Essentially enhanced and complex versions of integrity constraints

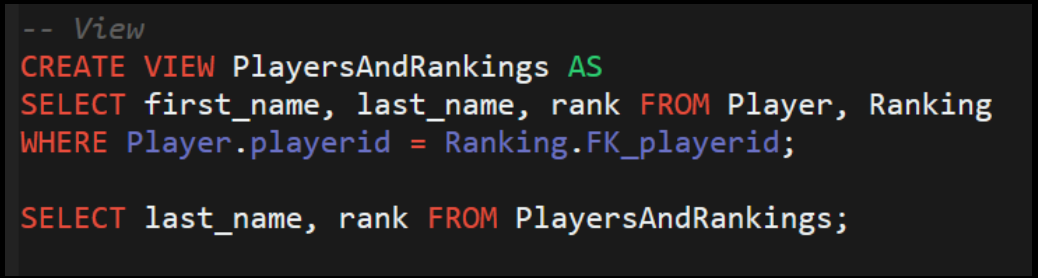
TRIGGER EXAMPLE



```
--SQLite trigger
CREATE TRIGGER createRank
AFTER INSERT ON Player

BEGIN
    INSERT INTO Ranking (points,rank,record, FK_playerid) VALUES (0,0,'0', NEW.playerid);
END;


--SQL Server trigger
CREATE TRIGGER createRank ON Player
AFTER INSERT
AS
    INSERT INTO Ranking (points,rank,record, FK_playerid) SELECT 0,0,'0', p.playerid
    FROM inserted p
GO
```

VIEW

- Views are virtual tables based on SELECT statements
- Any SELECT statement can be stored as a view for easier access and usage
- Views can be queries in the same fashion as regular tables
- Views can be updated if and only if
  - It is based on one table
  - Includes PK of the table
  - Has no GROUP BY, HAVING, DISTINCT clauses
  - Has no subqueries nor aggregation functions
  - In short, avoid trying to update views.



```sql
-- View
CREATE VIEW PlayersAndRankings AS
SELECT first_name, last_name, rank FROM Player, Ranking
WHERE Player.playerid = Ranking.FK_playerid;

SELECT last_name, rank FROM PlayersAndRankings;
```

LUT University