Item Market Database

Ryan Munger

Flow Of Contents

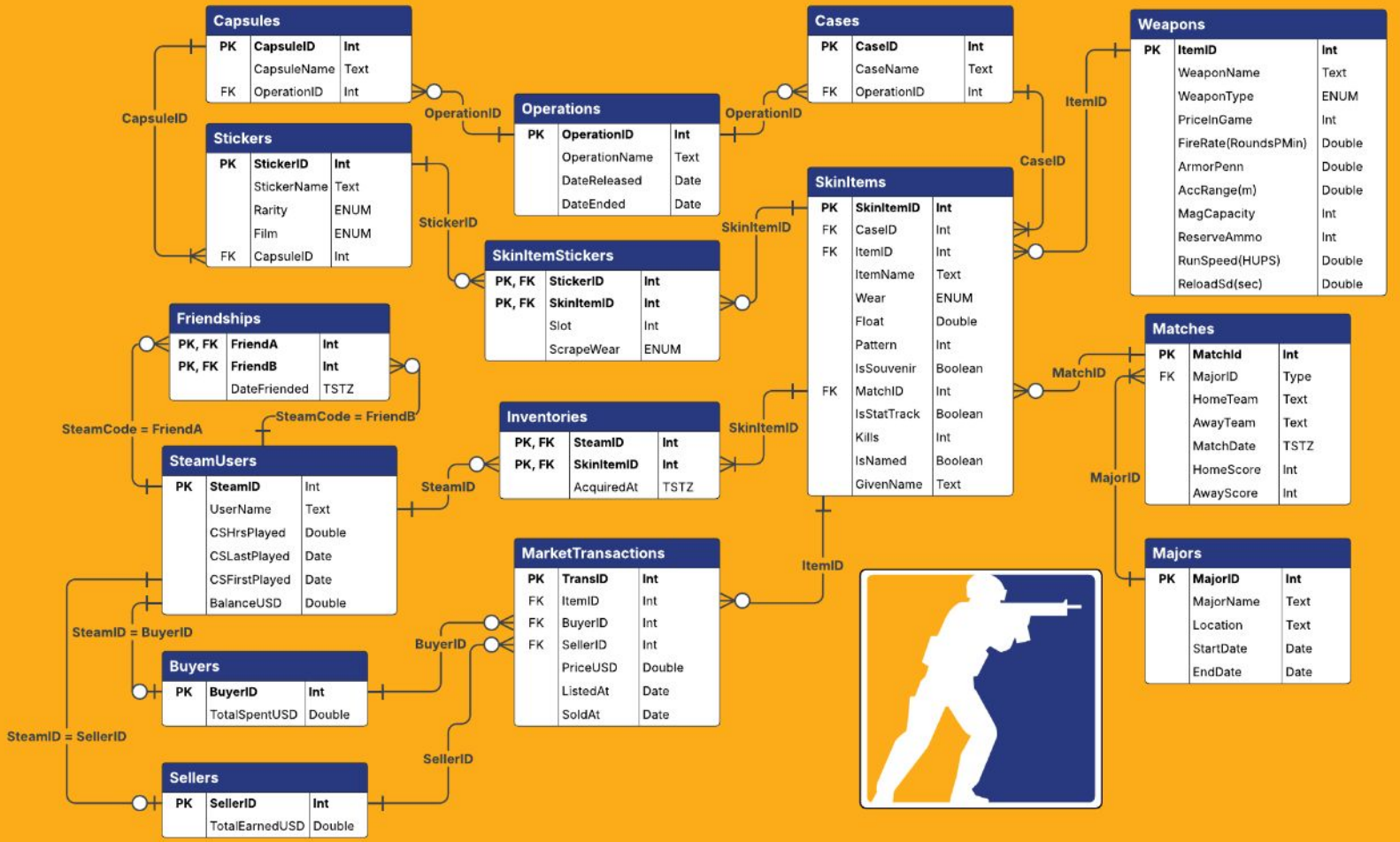
Executive Summary

This document provides an overview of the database schema designed for a CounterStrike2 (CS2) item marketplace. The database captures the relationships between various entities within the CS ecosystem, including operations, cases, capsules, stickers, majors, matches, weapons, skin items, users, and market transactions. This schema aims to facilitate efficient storage and retrieval of data related to CS items, their attributes, and their trading history. This document contains the respective create statements and documentation for each table as well as useful reports (queries), view definitions, stored procedures, triggers, and security features.

Objectives

The primary objectives of this database are to:

- **Organize CS Item Data:** Systematically structure data related to CS items (skins, stickers, etc.) and their characteristics.
- **Track Items:** Record the origin and history of items, including the cases or capsules they were obtained from.
- **Manage Market Transactions:** Store data on item sales, including buyer, seller, price, and timestamps.
- **Store User Information:** Maintain data on Steam users, including their inventories and trading activity.
- **Support Major Tournament Data:** Store information about CS Major tournaments, matches, and related skin items.
- **Enable Complex Queries:** Facilitate complex queries for data analysis, reporting, and application development.
- **Ensure Data Integrity:** Enforce data consistency and accuracy through constraints, data types, and relationships.


Capsules
PK CapsuleID Int
CapsuleName Text
FK OperationID Int
Stickers
PK StickerID Int
StickerName Text
Rarity ENUM
Film ENUM
FK CapsuleID Int
CapsuleID
OperationID
Operations
PK OperationID Int
OperationName Text
DateReleased Date
DateEnded Date
Cases
PK CaseID Int
CaseName Text
FK OperationID Int
Weapons
PK ItemID Int
WeaponName Text
WeaponType ENUM
PriceInGame Int
FireRate(RoundsPMin) Double
ArmorPenn Double
AccRange(m) Double
MagCapacity Int
ReserveAmmo Int
RunSpeed(HUPS) Double
ReloadSd(sec) Double
ItemID
CaseID
StickerID
SkinItemStickers
PK, FK StickerID Int
PK, FK SkinItemID Int
Slot Int
ScrapeWear ENUM
SkinItemID
SkinItems
PK SkinItemID Int
FK CaseID Int
FK ItemID Int
ItemName Text
Wear ENUM
Float Double
Pattern Int
IsSouvenir Boolean
FK MatchID Int
IsStatTrack Boolean
Kills Int
IsNamed Boolean
GivenName Text
Friendships
PK, FK FriendA Int
PK, FK FriendB Int
DateFriended TSTZ
SteamCode = FriendA
SteamCode = FriendB
Matches
PK MatchId Int
FK MajorID Type
HomeTeam Text
AwayTeam Text
MatchDate TSTZ
HomeScore Int
AwayScore Int
MatchID
MajorID
Inventories
PK, FK SteamID Int
PK, FK SkinItemID Int
AcquiredAt TSTZ
SteamUsers
PK SteamID Int
UserName Text
CSHrsPlayed Double
CSLastPlayed Date
CSFirstPlayed Date
BalanceUSD Double
SteamID
Majors
PK MajorID Int
MajorName Text
Location Text
StartDate Date
EndDate Date
MarketTransactions
PK TransID Int
FK ItemID Int
FK BuyerID Int
FK SellerID Int
PriceUSD Double
ListedAt Date
SoldAt Date
SteamID = BuyerID
BuyerID
Buyers
PK BuyerID Int
TotalSpentUSD Double
SteamID = SellerID
SellerID
Sellers
PK SellerID Int
TotalEarnedUSD Double

ERD

Operations Table

Purpose: Stores information on seasonal operations (in-game events).

Create Statement:

```
CREATE TABLE Operations (
    OperationID INT PRIMARY KEY,
    OperationName TEXT NOT NULL,
    DateReleased DATE NOT NULL
);
```

	operationid [PK] integer	operationname text	datereleased date
1	1	Operation Breakout	2014-07-01
2	2	Operation Vanguard	2014-11-11
3	3	Operation Bloodhound	2015-05-26
4	4	Operation Wildfire	2016-02-17
5	5	Operation Hydra	2017-05-23
6	6	Operation Shattered Web	2019-11-18
7	7	Operation Broken Fang	2020-12-03
8	8	Operation Riptide	2021-09-21
9	9	Operation Recoil	2022-07-01
10	10	Operation Anubis	2023-04-25

Functional Dependencies: OperationID → OperationName, DateReleased

Cases Table

Purpose: Stores information on CS cases (contain items and are opened by users).

Create Statement:

	caseid [PK] integer	casename text	operationid integer
1	101	Breakout Case	1
2	102	Vanguard Case	2
3	103	Bloodhound Case	3
4	104	Wildfire Case	4
5	105	Hydra Case	5
6	106	Shattered Web Case	6
7	107	Broken Fang Case	7
8	108	Riptide Case	8
9	109	Recoil Case	9
10	110	Anubis Collection Package	10
11	111	Operation Phoenix Weapon Case	[null]
12	112	Chroma Case	[null]
13	113	Shadow Case	[null]

```
CREATE TABLE Cases (
    CaseID INT PRIMARY KEY,
    CaseName TEXT NOT NULL,
    OperationID INT,
    FOREIGN KEY (OperationID) REFERENCES Operations(OperationID)
);
```

Functional Dependencies: CaseID → CaseName, OperationID

Capsules Table

Purpose: Stores information on CS capsules (contain stickers and are opened by users).

Create Statement:

	capsuleid [PK] integer	capsulename text	operationid integer
1	201	Breakout Capsule	1
2	202	Vanguard Capsule	2
3	203	Bloodhound Capsule	3
4	204	Wildfire Capsule	4
5	205	Hydra Capsule	5
6	206	Shattered Web Sticker Capsule	6
7	207	Broken Fang Sticker Capsule	7
8	208	Riptide Sticker Capsule	8
9	209	Recoil Sticker Capsule	9
10	210	Anubis Sticker Capsule	10
11	211	Katowice 2014 Challengers	[null]
12	212	Cologne 2014 Legends	[null]

```
CREATE TABLE Capsules (
    CapsuleID INT PRIMARY KEY,
    CapsuleName TEXT NOT NULL,
    OperationID INT,
    FOREIGN KEY (OperationID) REFERENCES Operations(OperationID)
);
```

Functional Dependencies: CapsuleID → CapsuleName, OperationID

Stickers Table

Purpose: Stores information about CS stickers (obtained from capsules and can be applied to weapon skins).

Create Statement:

```
CREATE TABLE Stickers (
    StickerID INT PRIMARY KEY,
    StickerName TEXT NOT NULL,
    Rarity TEXT NOT NULL CHECK (Rarity IN ('Base Grade', 'Medium Grade', 'High Grade', 'Remarkable', 'Exotic', 'Extraordinary', 'Contraband')),
    Film TEXT NOT NULL CHECK (Film IN ('Foil', 'Glitter', 'Gold', 'Holo', 'Lenticular', 'Paper')),
    CapsuleID INT,
    FOREIGN KEY (CapsuleID) REFERENCES Capsules(CapsuleID)
);
```

	stickerid [PK] integer	stickername text	rarity text	film text	capsuleid integer
1	301	IBP Holo	Extraordinary	Holo	211
2	302	Titan Holo	Extraordinary	Holo	211
3	303	Virtus.pro Foil	Remarkable	Foil	212
4	304	NaVi Paper	Base Grade	Paper	206
5	305	Astralis Gold	Exotic	Gold	207
6	306	MOUZ Glitter	High Grade	Glitter	208
7	307	9INE Holo	Remarkable	Holo	210
8	308	Complexity Gaming Paper	Medium Gr...	Paper	209
9	309	G2 Esports Foil	High Grade	Foil	207
10	310	ENCE Holo	Remarkable	Holo	208
11	311	FURIA Esports Gold	Exotic	Gold	209
12	312	Spirit Paper	Base Grade	Paper	210

Functional Dependencies: StickerID → StickerName, Rarity, Film, CapsuleID

Majors Table

Purpose: Stores information about CS major tournaments.

Create Statement:

```
CREATE TABLE Majors (
    MajorID INT PRIMARY KEY,
    MajorName TEXT NOT NULL,
    Location TEXT,
    StartDate DATE CHECK (StartDate >= '2012-01-01'), -- game release
    EndDate DATE,
    CHECK (EndDate >= StartDate)
);
```

Functional Dependencies:

MajorID → MajorName, Location, StartDate, EndDate

	majorid [PK] integer	majorname text	location text	startdate date	enddate date
1	401	EMS One Katowice 20...	Katowice, Poland	2014-03-13	2014-03-16
2	402	ESL One Cologne 2014	Cologne, Germany	2014-08-14	2014-08-17
3	403	DreamHack Winter 2014	Jönköping, Sweden	2014-11-26	2014-11-29
4	404	IEM Katowice 2015	Katowice, Poland	2015-03-12	2015-03-15
5	405	ESL One Cologne 2015	Cologne, Germany	2015-08-20	2015-08-23

Matches Table

Purpose: Stores information about matches within a major. Items obtained during a match become souvenirs associated with it.

Create Statement:

```
CREATE TABLE Matches (
    MatchID INT PRIMARY KEY,
    MajorID INT NOT NULL,
    HomeTeam TEXT NOT NULL,
    AwayTeam TEXT NOT NULL,
    MatchDate TIMESTAMP WITH TIME ZONE,
    HomeScore INT DEFAULT 0 CHECK (HomeScore >= 0),
    AwayScore INT DEFAULT 0 CHECK (AwayScore >= 0),
    FOREIGN KEY (MajorID) REFERENCES Majors(MajorID)
);
```

Functional Dependencies:

MatchID → MajorID, HomeTeam,

	matchid [PK] integer	majorid integer	hometeam text	awayteam text	matchdate timestamp with time zone	homescore integer	awayscore integer
1	501	401	Virtus.pro	Ninjas in Pyjamas	2014-03-16 11:00:00-04	2	1
2	502	402	Ninjas in Pyjamas	Fnatic	2014-08-17 13:30:00-04	0	2
3	503	403	LDLC	Ninjas in Pyjamas	2014-11-29 14:00:00-05	2	1
4	504	404	Fnatic	Ninjas in Pyjamas	2015-03-15 12:45:00-04	2	0
5	505	405	Fnatic	EnvyUs	2015-08-23 14:15:00-04	2	0
6	506	405	Astralis	G2 Esports	2015-08-22 10:00:00-04	1	2

Weapons Table

Purpose: Stores information on the weapons that skins can be obtained for.

Create Statement:

```
CREATE TABLE Weapons (
    ItemID INT PRIMARY KEY,
    WeaponName TEXT NOT NULL,
    WeaponType TEXT NOT NULL CHECK (WeaponType IN ('Pistol', 'SMG', 'Rifle', 'Sniper', 'Shotgun', 'Machinegun', 'Knife', 'Gloves')),
    PriceInGame INT CHECK (PriceInGame >= 0),
    FireRate DOUBLE PRECISION CHECK (FireRate > 0),
    ArmorPen DOUBLE PRECISION CHECK (ArmorPen BETWEEN 0 AND 100), -- percentage
    AccRange INT CHECK (AccRange >= 0),
    MagCapacity INT CHECK (MagCapacity >= 0),
    ReserveAmmo INT CHECK (ReserveAmmo >= 0),
    RunSpeed DOUBLE PRECISION CHECK (RunSpeed > 0),
    ReloadSd DOUBLE PRECISION CHECK (ReloadSd > 0)
);
```

Functional Dependencies:

ItemID → WeaponName, WeaponType, PriceInGame, FireRate, ArmorPen, AccRange, MagCapacity, ReserveAmmo, RunSpeed, ReloadSd

	itemid [PK] integer	weaponname text	weapontype text	priceingame integer	firerate double precision	armorpen double precision	accrange integer	magcapacity integer	reserveammo integer	runspeed double precision	reloadsd double precision
1	123	Compiler Tyrant	Shotgun	9999	10	100	100	25	50	250	2.3
2	601	AK-47	Rifle	2700	600	80	26	30	90	215	2.5
3	602	M4A4	Rifle	3100	667	80	27	30	90	225	3.1
4	603	AWP	Sniper	4750	41	97.5	69	10	20	200	3.6
5	604	Glock-18	Pistol	200	400	0	22	20	120	245	2.2
6	605	USP-S	Pistol	200	353	20	28	12	24	230	2.2
7	606	Desert Eagle	Pistol	700	267	90	16	7	35	230	2.2
8	607	MP9	SMG	1250	857	57	17	30	120	235	2.1
9	608	AWP	Sniper	4750	41	97.5	69	10	20	200	3.6
10	609	M4A1-S	Rifle	2900	600	80	29	20	40	225	2.8
11	610	Gut Knife	Knife	0	[null]	100	0	1	0	250	1
12	611	Sport Gloves	Gloves	0	[null]	0	0	1	0	250	1

SkinItems Table

Purpose: Stores information about skins for weapons.

```
CREATE TABLE SkinItems (
    SkinItemID INT PRIMARY KEY,
    CaseID INT,
    ItemID INT NOT NULL,
    ItemName TEXT NOT NULL,
    Wear TEXT NOT NULL CHECK (Wear IN ('Factory New', 'Minimal Wear', 'Field-Tested', 'Well-Worn', 'Battle-Scarred')),
    Float DOUBLE PRECISION NOT NULL CHECK (Float BETWEEN 0 AND 1),
    Pattern INT CHECK (Pattern >= 0),
    IsSouvenir BOOLEAN NOT NULL DEFAULT FALSE,
    MatchID INT,
    IsStatTrak BOOLEAN NOT NULL DEFAULT FALSE,
    Kills INT DEFAULT 0 CHECK (Kills >= 0),
    IsNamed BOOLEAN NOT NULL DEFAULT FALSE,
    GivenName TEXT,
    FOREIGN KEY (CaseID) REFERENCES Cases(CaseID),
    FOREIGN KEY (ItemID) REFERENCES Weapons(ItemID),
    FOREIGN KEY (MatchID) REFERENCES Matches(MatchID)
);
```

Functional Dependencies: SkinItemID → CaseID, ItemID, ItemName, Wear, Float, Pattern, IsSouvenir, MatchID, IsStatTrak, Kills, IsNamed, GivenName

Example SkinItems

	skinitemid [PK] integer	caseid integer	itemid integer	itemname text	wear text	float double precision	pattern integer	issouvenir boolean	matchid integer	isstattrak boolean	kills integer	isnamed boolean	givenname text
1	2112	101	123	Compiler Tyrant | Labouseur	Factory New	0.01	7	false	[null]	true	900	true	The Alanator
2	701	101	601	AK-47 | Redline	Field-Tested	0.28	42	false	[null]	false	0	false	[null]
3	702	101	607	MP9 | Hypnotic	Minimal Wear	0.11	17	false	[null]	true	15	false	[null]
4	703	111	603	AWP | Lightning Strike	Factory New	0.05	88	false	[null]	false	0	true	Zapatron
5	704	112	602	M4A4 | Asiimov	Well-Worn	0.41	12	false	[null]	true	55	true	Orion
6	705	106	605	USP-S | Kill Confirmed	Minimal Wear	0.18	5	false	[null]	false	0	false	[null]
7	706	107	609	M4A1-S | Printstream	Factory New	0.03	163	false	[null]	false	0	false	[null]
8	707	108	606	Desert Eagle | Ocean Drive	Field-Tested	0.35	22	false	[null]	true	28	false	[null]
9	708	104	610	Gut Knife | Doppler	Factory New	0.01	999	false	[null]	false	0	false	[null]
10	709	105	611	Hydra Gloves | Case Hardened	Minimal Wear	0.14	[null]	false	[null]	false	0	true	Retirement
11	710	102	601	AK-47 | Wasteland Rebel	Battle-Scarred	0.78	10	true	501	false	0	false	[null]
12	711	103	603	AWP | Man-o-war	Minimal Wear	0.15	33	true	502	true	112	true	Long Shot
13	712	104	602	M4A4 | Griffin	Field-Tested	0.31	7	false	[null]	false	0	false	[null]
14	713	105	607	MP9 | Wild Lily	Factory New	0.07	2	false	[null]	false	0	false	[null]
15	714	109	605	USP-S | Cortex	Minimal Wear	0.12	18	false	[null]	true	41	false	[null]
16	715	110	609	M4A1-S | Welcome to the Jungle	Field-Tested	0.25	55	false	[null]	false	0	false	[null]

SkinItemStickers Table

Purpose: Stores information about stickers applied to skin items.

Create Statement:

```
CREATE TABLE SkinItemStickers (
    StickerID INT UNIQUE, -- a sticker can only be applied to one item
    SkinItemID INT,
    Slot INT CHECK (Slot BETWEEN 0 AND 4),
    ScrapeWear TEXT CHECK (ScrapeWear IN ('Pristine', 'Scratched', 'Worn', 'Damaged')),
    PRIMARY KEY (StickerID, SkinItemID),
    FOREIGN KEY (SkinItemID) REFERENCES SkinItems(SkinItemID),
    FOREIGN KEY (StickerID) REFERENCES Stickers(StickerID)
);
```

	stickerid [PK] integer	skinitemid [PK] integer	slot integer	scrapewear text
1	301	703	0	Pristine
2	302	703	1	Scratched
3	304	701	0	Worn
4	305	704	2	Pristine
5	303	711	0	Pristine
6	306	702	1	Scratched
7	307	715	0	Pristine
8	308	712	3	Worn
9	309	706	0	Pristine
10	310	707	2	Scratched
11	311	710	1	Worn
12	312	714	4	Pristine

Functional Dependencies: StickerID, SkinItemID → Slot, ScrapeWear

SteamUsers Table

Purpose: Stores information about users that can collect items.

Create Statement:

```
CREATE TABLE SteamUsers (
    SteamID INT PRIMARY KEY,
    UserName TEXT NOT NULL,
    CSHrsPlayed DOUBLE PRECISION DEFAULT 0 CHECK (CSHrsPlayed >= 0),
    CSLastPlayed DATE,
    CSFirstPlayed DATE,
    BalanceUSD DOUBLE PRECISION DEFAULT 0
);
```

Functional Dependencies:

SteamID → UserName, CSHrsPlayed, CSLastPlayed, CSFirstPlayed, BalanceUSD

	steamid [PK] integer	username text	cshrsplayed double precision	cslastplayed date	csfirstplayed date	balanceusd double precision
1	007	database_god	2112	2025-04-03	2015-03-29	99999999
2	801	gabenfan123	5421.7	2025-04-23	2012-08-15	125.5
3	802	counterstrike_pro	12876.3	2025-04-24	2013-01-20	55.2
4	803	skin_collector	2345.9	2025-04-22	2016-05-01	1003.85
5	804	noob_player	150.2	2025-04-20	2024-11-10	5.99
6	805	trader_joe	876.5	2025-04-23	2019-03-01	2500
7	806	major_fan	3000.1	2025-04-21	2017-07-01	78.9

Inventories Table

Purpose: Stores information about what items belong to which users.

Create Statement:

```
CREATE TABLE Inventories (
    SteamID INT,
    SkinItemID INT UNIQUE, -- Only one person can own an item at a time
    AcquiredAt TIMESTAMP WITH TIME ZONE DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (SteamID, SkinItemID),
    FOREIGN KEY (SteamID) REFERENCES SteamUsers(SteamID),
    FOREIGN KEY (SkinItemID) REFERENCES SkinItems(SkinItemID)
);
```

	steamid [PK] integer	skinitemid [PK] integer	acquiredat timestamp with time zone
1	805	702	2025-04-12 12:45:00-04
2	802	704	2025-04-16 07:00:00-04
3	802	701	2025-04-11 06:30:00-04
4	805	710	2025-04-19 05:00:00-04
5	802	705	2025-04-21 09:15:00-04
6	805	711	2025-04-22 06:00:00-04
7	805	707	2025-04-23 04:45:00-04
8	802	703	[default]
9	803	706	2025-03-22 15:00:00-04
10	801	714	2025-04-15 17:00:00-04
11	802	715	2025-04-20 12:45:00-04

Functional Dependencies: SteamID, SkinItemID → AcquiredAt

Friendships Table

Purpose: Stores information about user friendships (undirected).

Create Statement:

```
CREATE TABLE Friendships (
    FriendA INT,
    FriendB INT,
    DateFriended TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (FriendA, FriendB),
    FOREIGN KEY (FriendA) REFERENCES SteamUsers(SteamID),
    FOREIGN KEY (FriendB) REFERENCES SteamUsers(SteamID),
    CHECK (FriendA != FriendB)
);
```

	frienda [PK] integer	friendb [PK] integer	datefriended timestamp with time zone
1	801	802	2018-06-01 07:30:00-04
2	801	803	2020-01-10 10:45:00-05
3	802	803	2021-05-22 05:00:00-04
4	804	801	2024-12-25 13:00:00-05
5	805	803	2022-08-01 10:00:00-04

Functional Dependencies: FriendA, FriendB → DateFriended

Buyers Table

Purpose: Stores information on users buying items.

Create Statement:

	buyerid [PK] integer	totalspentusd double precision
1	801	350.75
2	803	5200.1
3	804	15.5
4	806	112.99

```
CREATE TABLE Buyers (
    BuyerID INT PRIMARY KEY,
    TotalSpentUSD DOUBLE PRECISION DEFAULT 0 CHECK (TotalSpentUSD >= 0),
    FOREIGN KEY (BuyerID) REFERENCES SteamUsers(SteamID)
);
```

Functional Dependencies: BuyerID → TotalSpentUSD

Sellers Table

Purpose: Stores information on users selling items.

Create Statement:

```
CREATE TABLE Sellers (
    SellerID INT PRIMARY KEY,
    TotalEarnedUSD DOUBLE PRECISION DEFAULT 0 CHECK (TotalEarnedUSD >= 0),
    FOREIGN KEY (SellerID) REFERENCES SteamUsers(SteamID)
);
```

	sellerid [PK] integer	totalearnedusd double precision
1	802	1200.5
2	803	4850
3	805	2875.25

Functional Dependencies: SellerID → TotalEarnedUSD

MarketTransactions Table

```sql
CREATE TABLE MarketTransactions (
    TransID INT PRIMARY KEY,
    ItemID INT NOT NULL,
    BuyerID INT,
    SellerID INT NOT NULL,
    PriceUSD DOUBLE PRECISION NOT NULL CHECK (PriceUSD > 0),
    ListedAt TIMESTAMP WITH TIME ZONE NOT NULL DEFAULT CURRENT_TIMESTAMP,
    SoldAt TIMESTAMP WITH TIME ZONE CHECK (SoldAt >= ListedAt),
    FOREIGN KEY (ItemID) REFERENCES SkinItems(SkinItemID),
    FOREIGN KEY (BuyerID) REFERENCES Buyers(BuyerID),
    FOREIGN KEY (SellerID) REFERENCES Sellers(SellerID),
    CHECK (BuyerID != SellerID) -- selling to yourself?
);
```

Purpose: Stores information on item sale transactions.

	transid [PK] integer	itemid integer	buyerid integer	sellerid integer	priceusd double precision	listedat timestamp with time zone	soldat timestamp with time zone
1	901	701	801	802	45.5	2025-04-10 14:00:00-04	2025-04-11 06:30:00-04
2	902	702	801	805	120	2025-04-12 05:15:00-04	2025-04-12 12:45:00-04
3	903	704	803	802	850.25	2025-04-15 10:00:00-04	2025-04-16 07:00:00-04
4	904	710	803	805	675	2025-04-18 16:30:00-04	2025-04-19 05:00:00-04
5	905	705	804	802	12.75	2025-04-20 07:45:00-04	2025-04-21 09:15:00-04
6	906	711	803	805	1550	2025-04-21 12:00:00-04	2025-04-22 06:00:00-04
7	907	707	806	805	65.99	2025-04-22 15:30:00-04	2025-04-23 04:45:00-04
8	908	703	803	802	2100	2025-04-23 06:00:00-04	[null]

Functional Dependencies:

TransID → ItemID, BuyerID, SellerID, PriceUSD, ListedAt, SoldAt

Indexes

A variety of useful indexes are available to speed up data retrieval operations by allowing the database to quickly locate rows matching query conditions, especially on large tables.

```
-- Buyers & Sellers
CREATE INDEX idx_buyers_total_spent ON Buyers(TotalSpentUSD);
CREATE INDEX idx_sellers_total_earned ON Sellers(TotalEarnedUSD);

-- MarketTransactions
CREATE INDEX idx_markettransactions_buyerid ON MarketTransactions(BuyerID);
CREATE INDEX idx_markettransactions_sellerid ON MarketTransactions(SellerID);
CREATE INDEX idx_markettransactions_itemid ON MarketTransactions(ItemID);
CREATE INDEX idx_markettransactions_listedat ON MarketTransactions(ListedAt);
CREATE INDEX idx_markettransactions_soldat ON MarketTransactions(SoldAt);
```

```
-- Operations
CREATE INDEX idx_operations_date_released ON Operations(DateReleased);

-- Cases
CREATE INDEX idx_cases_operation_id ON Cases(OperationID);

-- Capsules
CREATE INDEX idx_capsules_operation_id ON Capsules(OperationID);

-- Stickers
CREATE INDEX idx_stickers_capsule_id ON Stickers(CapsuleID);
CREATE INDEX idx_stickers_rarity ON Stickers(Rarity);
CREATE INDEX idx_stickers_film ON Stickers(Film);

-- Majors
CREATE INDEX idx_majors_start_date ON Majors(StartDate);
CREATE INDEX idx_majors_end_date ON Majors(EndDate);

-- Matches
CREATE INDEX idx_matches_major_id ON Matches(MajorID);
CREATE INDEX idx_matches_match_date ON Matches(MatchDate);

-- Weapons
CREATE INDEX idx_weapons_weapon_type ON Weapons(WeaponType);

-- SkinItems
CREATE INDEX idx_skinitems_case_id ON SkinItems(CaseID);
CREATE INDEX idx_skinitems_item_id ON SkinItems(ItemID);
CREATE INDEX idx_skinitems_match_id ON SkinItems(MatchID);
CREATE INDEX idx_skinitems_float ON SkinItems(Float);
CREATE INDEX idx_skinitems_wear ON SkinItems(Wear);

-- SkinItemStickers
CREATE INDEX idx_skinitemstickers_skinitem_id ON SkinItemStickers(SkinItemID);
CREATE INDEX idx_skinitemstickers_slot ON SkinItemStickers(Slot);

-- Inventories
CREATE INDEX idx_inventories_steamid ON Inventories(SteamID);
CREATE INDEX idx_inventories_acquiredat ON Inventories(AcquiredAt);

-- Friendships
CREATE INDEX idx_friendships_friendb ON Friendships(FriendB);
CREATE INDEX idx_friendships_datefriended ON Friendships(DateFriended);
```

```
SELECT *
FROM pg_indexes
WHERE schemaname = 'public';
```

schemaname	tablename	indexname	tablespace	
public	operations	operations_pkey		CREATE UNIQUE INDEX operations_pkey ON public.operations USING btree (operationid)
public	cases	cases_pkey		CREATE UNIQUE INDEX cases_pkey ON public.cases USING btree (caseid)
public	capsules	capsules_pkey		CREATE UNIQUE INDEX capsules_pkey ON public.capsules USING btree (capsuleid)
public	stickers	stickers_pkey		CREATE UNIQUE INDEX stickers_pkey ON public.stickers USING btree (stickerid)
public	majors	majors_pkey		CREATE UNIQUE INDEX majors_pkey ON public.majors USING btree (majorid)
public	matches	matches_pkey		CREATE UNIQUE INDEX matches_pkey ON public.matches USING btree (matchid)
public	weapons	weapons_pkey		CREATE UNIQUE INDEX weapons_pkey ON public.weapons USING btree (itemid)
public	skinitems	skinitems_pkey		CREATE UNIQUE INDEX skinitems_pkey ON public.skinitems USING btree (skinitemid)
public	skinitemstickers	skinitemstickers_pkey		CREATE UNIQUE INDEX skinitemstickers_pkey ON public.skinitemstickers USING btree (stickerid, skinitemid)
public	skinitemstickers	skinitemstickers_stickerid_key		CREATE UNIQUE INDEX skinitemstickers_stickerid_key ON public.skinitemstickers USING btree (stickerid)
public	steamusers	steamusers_pkey		CREATE UNIQUE INDEX steamusers_pkey ON public.steamusers USING btree (steamid)
public	inventories	inventories_pkey		CREATE UNIQUE INDEX inventories_pkey ON public.inventories USING btree (steamid, skinitemid)
public	inventories	inventories_skinitemid_key		CREATE UNIQUE INDEX inventories_skinitemid_key ON public.inventories USING btree (skinitemid)
public	friendships	friendships_pkey		CREATE UNIQUE INDEX friendships_pkey ON public.friendships USING btree (frienda, friendb)
public	buyers	buyers_pkey		CREATE UNIQUE INDEX buyers_pkey ON public.buyers USING btree (buyerid)
public	sellers	sellers_pkey		CREATE UNIQUE INDEX sellers_pkey ON public.sellers USING btree (sellerid)
public	markettransactions	markettransactions_pkey		CREATE UNIQUE INDEX markettransactions_pkey ON public.markettransactions USING btree (transid)
public	operations	idx_operations_date_released		CREATE INDEX idx_operations_date_released ON public.operations USING btree (datereleased)
public	cases	idx_cases_operation_id		CREATE INDEX idx_cases_operation_id ON public.cases USING btree (operationid)
public	capsules	idx_capsules_operation_id		CREATE INDEX idx_capsules_operation_id ON public.capsules USING btree (operationid)
public	stickers	idx_stickers_capsule_id		CREATE INDEX idx_stickers_capsule_id ON public.stickers USING btree (capsuleid)
public	stickers	idx_stickers_rarity		CREATE INDEX idx_stickers_rarity ON public.stickers USING btree (rarity)
public	stickers	idx_stickers_film		CREATE INDEX idx_stickers_film ON public.stickers USING btree (film)
public	majors	idx_majors_start_date		CREATE INDEX idx_majors_start_date ON public.majors USING btree (startdate)
public	majors	idx_majors_end_date		CREATE INDEX idx_majors_end_date ON public.majors USING btree (enddate)
public	matches	idx_matches_major_id		CREATE INDEX idx_matches_major_id ON public.matches USING btree (majorid)
public	matches	idx_matches_match_date		CREATE INDEX idx_matches_match_date ON public.matches USING btree (matchdate)
public	weapons	idx_weapons_weapon_type		CREATE INDEX idx_weapons_weapon_type ON public.weapons USING btree (weapontype)
public	skinitems	idx_skinitems_case_id		CREATE INDEX idx_skinitems_case_id ON public.skinitems USING btree (caseid)
public	skinitems	idx_skinitems_item_id		CREATE INDEX idx_skinitems_item_id ON public.skinitems USING btree (itemid)
public	skinitems	idx_skinitems_match_id		CREATE INDEX idx_skinitems_match_id ON public.skinitems USING btree (matchid)
public	skinitems	idx_skinitems_float		CREATE INDEX idx_skinitems_float ON public.skinitems USING btree ("float")
public	skinitems	idx_skinitems_wear		CREATE INDEX idx_skinitems_wear ON public.skinitems USING btree (wear)
public	skinitemstickers	idx_skinitemstickers_skinitem_id		CREATE INDEX idx_skinitemstickers_skinitem_id ON public.skinitemstickers USING btree (skinitemid)
public	skinitemstickers	idx_skinitemstickers_slot		CREATE INDEX idx_skinitemstickers_slot ON public.skinitemstickers USING btree (slot)
public	inventories	idx_inventories_steamid		CREATE INDEX idx_inventories_steamid ON public.inventories USING btree (steamid)
public	inventories	idx_inventories_acquiredat		CREATE INDEX idx_inventories_acquiredat ON public.inventories USING btree (acquiredat)
public	friendships	idx_friendships_friendb		CREATE INDEX idx_friendships_friendb ON public.friendships USING btree (friendb)
public	friendships	idx_friendships_datefriended		CREATE INDEX idx_friendships_datefriended ON public.friendships USING btree (datefriended)

View Report:
Popular Skins

Purpose: This view finds the most popular skins being bought and sold on the market.

```sql
-- Popular skins by market volume
CREATE VIEW popular_skins AS
SELECT
    s.SkinItemID,
    s.ItemName,
    w.WeaponName,
    s.Wear,
    COUNT(mt.TransID) AS transaction_count,
    AVG(mt.PriceUSD) AS average_price,
    SUM(mt.PriceUSD) AS total_volume
FROM SkinItems s
    INNER JOIN Weapons w ON s.ItemID = w.ItemID
    INNER JOIN MarketTransactions mt ON s.SkinItemID = mt.ItemID
GROUP BY s.SkinItemID, s.ItemName, w.WeaponName, s.Wear
ORDER BY transaction_count DESC;

SELECT * FROM popular_skins;
```

```sql
SELECT * FROM popular_skins;
```

	skinitemid integer	itemname text	weaponname text	wear text	transaction_count bigint	average_price double precision	total_volume double precision
1	701	AK-47 | Redline	AK-47	Field-Tested	1	45.5	45.5
2	704	M4A4 | Asiimov	M4A4	Well-Worn	1	850.25	850.25
3	702	MP9 | Hypnotic	MP9	Minimal Wear	1	120	120
4	710	AK-47 | Wasteland Rebel	AK-47	Battle-Scarred	1	675	675
5	711	AWP | Man-o-war	AWP	Minimal Wear	1	1550	1550
6	707	Desert Eagle | Ocean Drive	Desert Eagle	Field-Tested	1	65.99	65.99
7	705	USP-S | Kill Confirmed	USP-S	Minimal Wear	1	12.75	12.75
8	703	AWP | Lightning Strike	AWP	Factory New	1	2100	2100

View Report:
Top Market Users

Purpose: This view shows the top market buyers and sellers.

```sql
-- Top sellers and buyers
CREATE VIEW top_market_users AS
SELECT
    'Seller' AS UserType,
    s.SellerID AS UserID,
    u.UserName,
    s.TotalEarnedUSD AS TotalEarnedUSD,
    NULL AS TotalSpentUSD -- NULL col for buyers
FROM Sellers s
    INNER JOIN SteamUsers u ON s.SellerID = u.SteamID
UNION ALL
SELECT
    'Buyer' AS UserType,
    b.BuyerID AS UserID,
    u.UserName,
    NULL AS TotalEarnedUSD, -- NULL for buyers
    b.TotalSpentUSD AS TotalSpentUSD
FROM Buyers b
    INNER JOIN SteamUsers u ON b.BuyerID = u.SteamID
LIMIT 10;
```

```sql
SELECT * FROM top_market_users;
```

	usertype text	userid integer	username text	totalearnedusd double precision	totalspentusd double precision
1	Seller	802	counterstrike_pro	1200.5	[null]
2	Seller	803	skin_collector	4850	[null]
3	Seller	805	trader_joe	2875.25	[null]
4	Buyer	801	gabenfan123	[null]	350.75
5	Buyer	803	skin_collector	[null]	5200.1
6	Buyer	804	noob_player	[null]	15.5
7	Buyer	806	major_fan	[null]	112.99

View Report: Common SkinItem Sticker Combos

Purpose: This view shows the user the most common sticker and item skin combinations!

```
-- Sticker combinations popularity
CREATE VIEW common_sticker_skin_combos AS
SELECT
    s.StickerName,
    si.ItemName AS SkinName,
    COUNT(*) AS CombinationCount
FROM SkinItemStickers sis
    INNER JOIN Stickers s ON sis.StickerID = s.StickerID
    INNER JOIN SkinItems si ON sis.SkinItemID = si.SkinItemID
GROUP BY s.StickerName, si.ItemName
ORDER BY CombinationCount DESC
LIMIT 10;
```

```
SELECT * FROM common_sticker_skin_combos;
```

	stickername text	skinname text	combinationcount bigint
1	9INE Holo	M4A1-S | Welcome to the Jungle	1
2	MOUZ Glitter	MP9 | Hypnotic	1
3	Virtus.pro Foil	AWP | Man-o-war	1
4	G2 Esports Foil	M4A1-S | Printstream	1
5	IBP Holo	AWP | Lightning Strike	1
6	Titan Holo	AWP | Lightning Strike	1
7	ENCE Holo	Desert Eagle | Ocean Drive	1
8	Complexity Gaming Paper	M4A4 | Griffin	1
9	Astralis Gold	M4A4 | Asiimov	1
10	NaVi Paper	AK-47 | Redline	1

View Report:

Average StatTrak Kills By Weapon Type

Purpose: This view details the average amount of StatTrak™ kills for each weapon, regardless of skin.

```
SELECT * FROM avg_stattrak_kills_by_weapon;
```

	weaponname text	averagekills numeric
1	M4A4	55.0000000000000000
2	MP9	15.0000000000000000
3	Desert Eagle	28.0000000000000000
4	Compiler Tyrant	900.0000000000000000
5	USP-S	41.0000000000000000
6	AWP	112.0000000000000000

```sql
-- Average StatTrak weapon performance
CREATE VIEW avg_stattrak_kills_by_weapon AS
SELECT
    w.WeaponName,
    AVG(s.Kills) AS AverageKills
FROM  Weapons w
    INNER JOIN  SkinItems s ON w.ItemID = s.ItemID
WHERE s.IsStatTrak = TRUE
GROUP BY w.WeaponName;
```

View Report: Daily Transactions

Purpose: This view shows us how strong the item market is each day in the last 30 days.

```
CREATE VIEW daily_transactions AS
SELECT
    DATE_TRUNC('day', mt.SoldAt)::DATE AS sale_date,
    COUNT(*) AS daily_trans,
    SUM(mt.PriceUSD) AS daily_vol,
    AVG(mt.PriceUSD) AS avg_price,
    MAX(mt.PriceUSD) AS highest_sale,
    COUNT(DISTINCT mt.BuyerID) AS unique_buyers,
    COUNT(DISTINCT mt.SellerID) AS unique_sellers
FROM MarketTransactions mt
WHERE mt.SoldAt >= CURRENT_DATE - INTERVAL '30 days'
GROUP BY DATE_TRUNC('day', mt.SoldAt)::DATE
ORDER BY sale_date DESC;
```

```
SELECT * FROM daily_transactions;
```

	sale_date date	daily_trans bigint	daily_vol double precision	avg_price double precision	highest_sale double precision	unique_buyers bigint	unique_sellers bigint
1	2025-04-23	1	65.99	65.99	65.99	1	1
2	2025-04-22	1	1550	1550	1550	1	1
3	2025-04-21	1	12.75	12.75	12.75	1	1
4	2025-04-19	1	675	675	675	1	1
5	2025-04-16	1	850.25	850.25	850.25	1	1
6	2025-04-12	1	120	120	120	1	1
7	2025-04-11	1	45.5	45.5	45.5	1	1

View Report: Top Weapons for Skins

Purpose: This view shows how many skins are available for each weapon.

```
SELECT * FROM count_weapon_skins;
```

	weaponname text	skin_count bigint
1	AK-47	2
2	M4A4	2
3	MP9	2
4	USP-S	2
5	M4A1-S	2
6	AWP	2
7	Sport Gloves	1
8	Gut Knife	1
9	Compiler Tyrant	1
10	Desert Eagle	1

```
CREATE VIEW count_weapon_skins AS
SELECT
    w.WeaponName,
    COUNT(si.SkinItemID) AS skin_count
FROM Weapons w
    INNER JOIN SkinItems si ON w.ItemID = si.ItemID
GROUP BY  w.WeaponName
ORDER BY skin_count DESC;
```

View Report: Average Case Item Values

Purpose: This view shows the average value of items from each case.

```
CREATE VIEW avg_case_values AS
SELECT
    c.CaseName,
    COUNT(si.SkinItemID) AS items_in_case,
    AVG(mt.PriceUSD) AS avg_price
FROM Cases c
    INNER JOIN SkinItems si ON c.CaseID = si.CaseID
    INNER JOIN MarketTransactions mt ON si.SkinItemID = mt.ItemID
GROUP BY c.CaseName
ORDER BY avg_price DESC;
```

```
SELECT * FROM avg_case_values;
```

	casename text	items_in_case bigint	avg_price double precision
1	Operation Phoenix Weapon Case	1	2100
2	Bloodhound Case	1	1550
3	Chroma Case	1	850.25
4	Vanguard Case	1	675
5	Breakout Case	2	82.75
6	Riptide Case	1	65.99
7	Shattered Web Case	1	12.75

View Report: Operation Metrics

Purpose: This view shows the performance metrics for each operation. Metrics include transactions per day, unique skins, market volume, and cases released.

```
CREATE VIEW operation_metrics AS
SELECT
    o.OperationName AS Op_Name,
    o.DateReleased AS Date_Release,
    o.DateEnded AS Date_End,
    COUNT(DISTINCT c.CaseID) AS cases_released,
    COUNT(DISTINCT si.SkinItemID) AS unique_skins,
    COUNT(DISTINCT mt.TransID) AS total_transactions,
    SUM(mt.PriceUSD) AS total_market_volume,
    SUM(mt.PriceUSD) / COUNT(DISTINCT mt.TransID) AS avg_transaction_value,
    COUNT(DISTINCT mt.TransID) /  (o.dateEnded - o.dateReleased) AS transactions_per_day
FROM Operations o
    LEFT OUTER JOIN Cases c ON o.OperationID = c.OperationID
    LEFT OUTER JOIN SkinItems si ON c.CaseID = si.CaseID
    LEFT OUTER JOIN MarketTransactions mt ON si.SkinItemID = mt.ItemID
GROUP BY o.OperationID, o.OperationName, o.DateReleased
ORDER BY o.DateReleased DESC;
```

```
SELECT * FROM operation_metrics;
```

	op_name text	date_release date	date_end date	cases_released bigint	unique_skins bigint	total_transactions bigint	total_market_volume double precision	avg_transaction_value double precision	transactions_per_day bigint
1	Operation Anubis	2023-04-25	2023-10-02	1	1	0	[null]	[null]	0
2	Operation Recoil	2022-07-01	2022-11-16	1	1	0	[null]	[null]	0
3	Operation Riptide	2021-09-21	2022-02-22	1	1	1	65.99	65.99	0
4	Operation Broken Fang	2020-12-03	2021-05-03	1	1	0	[null]	[null]	0
5	Operation Shattered Web	2019-11-18	2020-03-31	1	1	1	12.75	12.75	0
6	Operation Hydra	2017-05-23	2017-11-13	1	2	0	[null]	[null]	0
7	Operation Wildfire	2016-02-17	2016-07-15	1	2	0	[null]	[null]	0
8	Operation Bloodhound	2015-05-26	2015-09-30	1	1	1	1550	1550	0
9	Operation Vanguard	2014-11-11	2015-03-31	1	1	1	675	675	0
10	Operation Breakout	2014-07-01	2014-10-02	1	3	2	165.5	82.75	0

Stored Procedure:
Get_User_Items

Purpose: Provided the id of a user, return all of the items in that user's inventory.

```
CREATE OR REPLACE FUNCTION get_user_items (steam_id INT)
RETURNS TABLE (
    ItemName TEXT,
    FloatVal DOUBLE PRECISION,
    Wear TEXT,
    IsStatTrak BOOLEAN,
    IsSouvenir BOOLEAN,
    GivenName  TEXT
) AS
$$
BEGIN
    RETURN QUERY
    SELECT
        si.ItemName,
        si.Float,
        si.Wear,
        si.IsStatTrak,
        si.IsSouvenir,
        si.GivenName
    FROM Inventories i
        INNER JOIN SkinItems si ON i.SkinItemID = si.SkinItemID
    WHERE i.SteamID = steam_id
    ORDER BY si.ItemName;
END;
$$
LANGUAGE plpgsql;
```

```
SELECT * FROM get_user_items(803);
```

	itemname text	floatval double precision	wear text	isstattrak boolean	issouvenir boolean	givenname text
1	Desert Eagle | Ocean Drive	0.35	Field-Tested	true	false	[null]
2	M4A1-S | Printstream	0.03	Factory N...	false	false	[null]

Stored Procedure:
Case_Opening_Stats

Purpose: Provided the name of a case, this function determines how often certain drops are obtained.

```
-- Case opening stats - shows most common weapons and wear ratings for
CREATE OR REPLACE FUNCTION case_opening_stats(case_name TEXT)
RETURNS TABLE (
    weapon_name TEXT,
    skin_name TEXT,
    wear_condition TEXT,
    drop_count BIGINT,
    drop_percentage NUMERIC(5,2)
) AS
$$
DECLARE
    total_drops BIGINT;
BEGIN
    -- get num of drops from case
    SELECT COUNT(*) INTO total_drops
    FROM Cases c
        INNER JOIN SkinItems s ON c.CaseID = s.CaseID
    WHERE c.CaseName = case_name;

    -- get stats
    RETURN QUERY
    SELECT
        w.WeaponName,
        s.ItemName,
        s.Wear,
        COUNT(*) AS drop_count,
        ROUND((COUNT(*) * 100.0 / total_drops), 2) AS drop_percentage
    FROM Cases c
        INNER JOIN SkinItems s ON c.CaseID = s.CaseID
        INNER JOIN Weapons w ON s.ItemID = w.ItemID
    WHERE c.CaseName = case_name
    GROUP BY w.WeaponName, s.ItemName, s.Wear
    ORDER BY drop_count DESC;
END;
$$
LANGUAGE plpgsql;
```

```
SELECT * FROM case_opening_stats('Breakout Case');
```

	weapon_name text	skin_name text	wear_condition text	drop_count bigint	drop_percentage numeric
1	AK-47	AK-47 | Redline	Field-Tested	1	33.33
2	Compiler Tyrant	Compiler Tyrant | Labouseur	Factory New	1	33.33
3	MP9	MP9 | Hypnotic	Minimal Wear	1	33.33

Stored Procedure:
Souvenir_Items_By_Major

Purpose: Provided a major name, this function finds all souvenir items dropped during it, their owners, their values, and more!

```sql
CREATE OR REPLACE FUNCTION souvenir_items_by_major(major_name_param TEXT)
RETURNS TABLE (
    major_name TEXT,
    major_location TEXT,
    match_description TEXT,
    weapon_name TEXT,
    skin_name TEXT,
    wear_condition TEXT,
    float_value DOUBLE PRECISION,
    owner_name TEXT,
    last_sold_price DOUBLE PRECISION,
    estimated_value DOUBLE PRECISION
) AS
$$
DECLARE
    major_id_var INT;
BEGIN
    -- get the major id
    SELECT MajorID INTO major_id_var
    FROM Majors
    WHERE MajorName = major_name_param;

    -- get souvenir items, their owners, price history, etc
    RETURN QUERY
    SELECT
        m.MajorName,
        m.Location,
        match.HomeTeam || ' vs ' || match.AwayTeam AS match_description,
        w.WeaponName,
        si.ItemName,
        si.Wear,
        si.Float,
        COALESCE(u.UserName, 'Unknown') AS owner_name,
        MAX(mt.PriceUSD) AS last_sold_price,
        COALESCE(AVG(mt.PriceUSD), 0) AS estimated_value
    FROM Majors m
        INNER JOIN Matches match ON m.MajorID = match.MajorID
        INNER JOIN SkinItems si ON match.MatchID = si.MatchID AND si.IsSouvenir = TRUE
        INNER JOIN Weapons w ON si.ItemID = w.ItemID
        LEFT JOIN Inventories inv ON si.SkinItemID = inv.SkinItemID
        LEFT JOIN SteamUsers u ON inv.SteamID = u.SteamID
        LEFT JOIN MarketTransactions mt ON si.SkinItemID = mt.ItemID
    WHERE m.MajorID = major_id_var
    GROUP BY
        m.MajorName, m.Location, match_description, w.WeaponName,
        si.ItemName, si.Wear, si.Float, owner_name
    ORDER BY estimated_value DESC;
END;
$$ LANGUAGE plpgsql;
```

```sql
SELECT * FROM souvenir_items_by_major('EMS One Katowice 2014');
```

	major_name text	major_location text	match_description text	weapon_name text	skin_name text	wear_condition text	float_value double precision	owner_name text	last_sold_price double precision	estimated_value double precision
1	EMS One Katowice 2014	Katowice, Poland	Virtus.pro vs Ninjas in Pyjamas	AK-47	AK-47 | Wasteland Rebel	Battle-Scarred	0.78	Unknown	675	675

Stored Procedure:
Get Market Transactions

Purpose: Provided an item name and wear, this function finds all of the market listings and sales of the item.

```
SELECT * FROM get_market_transactions('AK-47 | Redline', 'Field-Tested');
```

	itemname text	wear text	transid integer	priceusd double precision	soldatdate date
1	AK-47 | Redline	Field-Tested	901	45.5	2025-04-11

```
CREATE OR REPLACE FUNCTION get_market_transactions(
    item_name_param TEXT,
    wear_param TEXT
)
RETURNS TABLE (
    ItemName TEXT,
    Wear TEXT,
    TransID INT,
    PriceUSD DOUBLE PRECISION,
    SoldAtDate DATE
) AS
$$
BEGIN
    RETURN QUERY
    SELECT
        item_name_param AS ItemName,
        wear_param AS Wear,
        mt.TransID,
        mt.PriceUSD,
        DATE_TRUNC('day', mt.SoldAt)::DATE
    FROM
        MarketTransactions mt
    INNER JOIN
        SkinItems si ON mt.ItemID = si.SkinItemID
    WHERE
        si.ItemName = item_name_param
        AND si.Wear = wear_param
    ORDER BY DATE_TRUNC('day', mt.SoldAt)::DATE DESC
    LIMIT 20;
END;
$$ LANGUAGE plpgsql;
```

Trigger: Prevent Duplicate Friendships

Purpose: This function is triggered every time there is an insertion into the friendships table. Since the friendship system is undirected, we do not need a FriendB → FriendA entry if there is already a FriendA → FriendB entry.

```
INSERT INTO Friendships (FriendA, FriendB, DateFriended) VALUES
(802, 801, '2015-08-22 14:00:00+00');
```

```
ERROR:  Friendship already exists in the opposite direction.
CONTEXT:  PL/pgSQL function prevent_duplicate_friendships() line 8 at RAISE
```

```
CREATE OR REPLACE FUNCTION prevent_duplicate_friendships()
RETURNS TRIGGER AS
$$
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Friendships
        WHERE (FriendA = NEW.FriendB AND FriendB = NEW.FriendA)
    ) THEN
        RAISE EXCEPTION 'Friendship already exists in the opposite direction.';
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER check_duplicate_friendships
BEFORE INSERT ON Friendships
FOR EACH ROW
EXECUTE FUNCTION prevent_duplicate_friendships();
```

Trigger: Confirm Seller Ownership

```
INSERT INTO MarketTransactions (TransID, ItemID, BuyerID, SellerID, PriceUSD, ListedAt, SoldAt) VALUES
(901, 701, NULL, 802, 45.50, '2025-04-10 18:00:00+00', NULL);
```

```
ERROR:  Item 701 is not in the inventory of Seller 802.
CONTEXT:  PL/pgSQL function ensure_ownership_before_listing() line 8 at RAISE
```

```
CREATE OR REPLACE FUNCTION ensure_ownership_before_listing()
RETURNS TRIGGER AS
$$
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM Inventories
        WHERE SteamID = NEW.SellerID AND SkinItemID = NEW.ItemID
    ) THEN
        RAISE EXCEPTION 'Item % is not in the inventory of Seller %.', NEW.ItemID, NEW.SellerID;
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER check_ownership
BEFORE INSERT ON MarketTransactions
FOR EACH ROW
EXECUTE FUNCTION ensure_ownership_before_listing();
```

Purpose: This function is triggered when a seller lists an item and ensures that the item is actually in the seller's inventory.

Trigger:

Transfer Item Ownership

```sql
SELECT * FROM Inventories; --

UPDATE MarketTransactions
SET BuyerID = 007,
    SoldAt = CURRENT_TIMESTAMP
WHERE TransID = 908;

SELECT * FROM Inventories; --
```

←Before | After→

	steamid [PK] integer	skinitemid [PK] integer	acquiredat timestamp with time zone
1	805	702	2025-04-12 12:45:00-04
2	802	704	2025-04-16 07:00:00-04
3	802	701	2025-04-11 06:30:00-04
4	805	710	2025-04-19 05:00:00-04
5	802	705	2025-04-21 09:15:00-04
6	805	711	2025-04-22 06:00:00-04
7	805	707	2025-04-23 04:45:00-04
8	802	703	[default]
9	803	706	2025-03-22 15:00:00-04
10	801	714	2025-04-15 17:00:00-04
11	802	715	2025-04-20 12:45:00-04

	steamid [PK] integer	skinitemid [PK] integer	acquiredat timestamp with time zone
1	805	702	2025-04-12 12:45:00-04
2	802	704	2025-04-16 07:00:00-04
3	802	701	2025-04-11 06:30:00-04
4	805	710	2025-04-19 05:00:00-04
5	802	705	2025-04-21 09:15:00-04
6	805	711	2025-04-22 06:00:00-04
7	805	707	2025-04-23 04:45:00-04
8	803	706	2025-03-22 15:00:00-04
9	801	714	2025-04-15 17:00:00-04
10	802	715	2025-04-20 12:45:00-04
11	7	703	2025-04-24 17:16:46.845109-04

```sql
CREATE OR REPLACE FUNCTION transfer_ownership()
RETURNS TRIGGER AS
$$
BEGIN
    -- Only run if the transaction has a buyer and a sale time
    -- No inventory exchange for an unsold listing
    IF NEW.BuyerID IS NOT NULL AND NEW.SoldAt IS NOT NULL THEN
        -- Remove from seller's inventory
        DELETE FROM Inventories
        WHERE SteamID = OLD.SellerID AND SkinItemID = OLD.ItemID;

        -- Insert into buyer's inventory
        INSERT INTO Inventories (SteamID, SkinItemID, AcquiredAt)
        VALUES (NEW.BuyerID, NEW.ItemID, NEW.SoldAt);
    END IF;
    RETURN NEW;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_inventories_trigger ON MarketTransactions;
CREATE TRIGGER update_inventories_trigger
AFTER INSERT OR UPDATE ON MarketTransactions
FOR EACH ROW
WHEN (NEW.BuyerID IS NOT NULL AND NEW.SoldAt IS NOT NULL)
EXECUTE FUNCTION transfer_ownership();
```

Purpose: This function is triggered when a transaction is inserted or updated. It transfers the ownership of an item from the seller's inventory to the buyer's inventory on transaction completion.

Trigger: Transfer Funds

```
CREATE OR REPLACE FUNCTION transfer_balance()
RETURNS TRIGGER AS
$$
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM SteamUsers
        WHERE SteamID = NEW.BuyerID AND BalanceUSD >= NEW.PriceUSD
    ) THEN
        RAISE EXCEPTION 'Buyer % does not have sufficient funds to purchase Item % (Price: %USD, Balance: %USD).',
                        NEW.BuyerID, NEW.ItemID, NEW.PriceUSD, (SELECT BalanceUSD FROM SteamUsers WHERE SteamID = NEW.BuyerID);
    END IF;

    -- Add the purchase amount minus steam fee to the seller's balance
    UPDATE SteamUsers
    SET BalanceUSD = BalanceUSD + ((NEW.PriceUSD / 1.15) - 0.01)
    WHERE SteamID = NEW.SellerID;

    -- Remove funds from buyer
    UPDATE SteamUsers
    SET BalanceUSD = BalanceUSD - NEW.PriceUSD
    WHERE SteamID = NEW.BuyerID;

    -- Update total earned
    UPDATE Sellers
    SET TotalEarnedUSD = TotalEarnedUSD + ((NEW.PriceUSD / 1.15) - 0.01)
    WHERE SellerID = NEW.SellerID;

    -- Update totalSpent
    UPDATE Buyers
    SET TotalSpentUSD = TotalSpentUSD + NEW.PriceUSD
    WHERE BuyerID = NEW.BuyerID;

    RETURN NEW;
END;
$$
LANGUAGE plpgsql;
```

```
CREATE TRIGGER transfer_funds
BEFORE INSERT OR UPDATE ON MarketTransactions
FOR EACH ROW
EXECUTE FUNCTION transfer_balance();
```

```
ERROR:  Buyer 801 does not have sufficient funds to purchase Item 707 (Price: 99999USD, Balance: 125.5USD).
CONTEXT:  PL/pgSQL function transfer_balance() line 8 at RAISE
INSERT INTO MarketTransactions (TransID, ItemID, BuyerID, SellerID, PriceUSD, ListedAt, SoldAt) VALUES
(909, 707, 801, 805, 10, '2025-04-10 18:00:00+00', '2025-04-11 10:30:00+00');
```

←Before | After→

	steamid [PK] integer	username text	balanceusd double precision
1	7	database_god	99999999
2	801	gabenfan123	125.5
3	802	counterstrike_pro	55.2
4	803	skin_collector	1003.85
5	804	noob_player	5.99
6	805	trader_joe	2500
7	806	major_fan	78.9

	steamid [PK] integer	username text	balanceusd double precision
1	7	database_god	99999999
2	801	gabenfan123	115.5
3	802	counterstrike_pro	55.2
4	803	skin_collector	1003.85
5	804	noob_player	5.99
6	805	trader_joe	2508.6856521739132
7	806	major_fan	78.9

Purpose: This function runs when a transaction is inserted or updated and ensures that the buyer has enough balance to purchase the item. It then transfers the funds properly as well as updates buyer totalSpent and seller totalEarned!

Security

Groups:

- Admins
- Application
- Analysts

Details

The security of this database has admin users, an application service user, and an analyst user.

The admin has full access to the entire database and schema.

The application service account can read and write data, but not alter the schema.

The analysts can only read data and utilize stored procedures.

Connections are limited for security.

```
-- basic roles
CREATE ROLE cs_admins WITH NOLOGIN;
CREATE ROLE cs_app WITH NOLOGIN;
CREATE ROLE cs_readonly WITH NOLOGIN;

-- Create user accounts and assign to groups
-- Very secure passwords !
-- Administrators (full access)
CREATE USER admin1 WITH PASSWORD 'admin_pass1' IN ROLE cs_admins;
-- Application service account
CREATE USER app_service WITH PASSWORD 'app_pass1' IN ROLE cs_app;
-- analysts (read only)
CREATE USER analyst1 WITH PASSWORD 'analyst_pass1' IN ROLE cs_readonly;

-- Grant permissions
-- Admin - full access
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO cs_admins;
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA public TO cs_admins;
GRANT ALL PRIVILEGES ON ALL FUNCTIONS IN SCHEMA public TO cs_admins;
-- App service cannot modify the schema but it can write
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA public TO cs_app;
GRANT USAGE ON ALL SEQUENCES IN SCHEMA public TO cs_app;
GRANT EXECUTE ON ALL FUNCTIONS IN SCHEMA public TO cs_app;
-- analysts just get select and procs
GRANT SELECT ON ALL TABLES IN SCHEMA public TO cs_readonly;
GRANT EXECUTE ON FUNCTION case_opening_stats(TEXT) TO cs_readonly;
GRANT EXECUTE ON FUNCTION market_price_trends(TEXT) TO cs_readonly;
GRANT EXECUTE ON FUNCTION souvenir_items_by_major(TEXT) TO cs_readonly;

-- Create connection limits for security - saw while researching syntax - pretty neat
ALTER ROLE cs_support CONNECTION LIMIT 10;
ALTER ROLE cs_readonly CONNECTION LIMIT 5;
ALTER ROLE cs_app CONNECTION LIMIT 50;
```

Implementation Notes

Friendships are undirected.

Operations are in-game events that contain cases, capsules, etc. Not all cases and capsules need to be part of an operation.

Items obtained from a case during a major match will become a souvenir item.

There is a steam fee whenever an item is sold. Items can be named by players. Items can be StatTrak™, meaning that the kills achieved with the weapon are counted. Items have varying levels of wear on them, indicating damage to the item. The float is simply a more precise description of this wear as opposed to the wear name. The same skin for the same item may also come in several patterns.

Stickers can be scraped off of items to varying degrees. A sticker's 'Film' refers to its finish, for example, a metallic foil. Stickers can be applied to items at preset locations, known as slots. Some items have more slots than others.

Known Problems

This database has a small set of known flaws:

The ‘Location’ field of the Majors table can be further broken down for normalization purposes.

The MarketTransactions Table may not scale well to a busy market system.

User inventories cannot hold stickers, so it is unclear how users could obtain them or apply them to their items.

The SkinItemStickers table does not prevent the same item from having over 4 stickers applied to it. It is also possible to apply multiple stickers in the same slot. Both of these scenarios are erroneous. Triggers would fix these issues.

Future Enhancements

There are several features in the real CounterStrike item & market system that could be added to this database implementation.

Some sticker capsules are associated with majors similar to souvenir skins. This functionality could be added with a foreign key to the majors table.

A user's inventory can contain more than just weapon skins. It can also hold stickers, capsules, cases, case keys, name tags, sprays, music kits, etc!

Item trades between users can be implemented.

Users can place 'buy orders' to buy a specific quantity of a specific item at a specific price.

Thank You