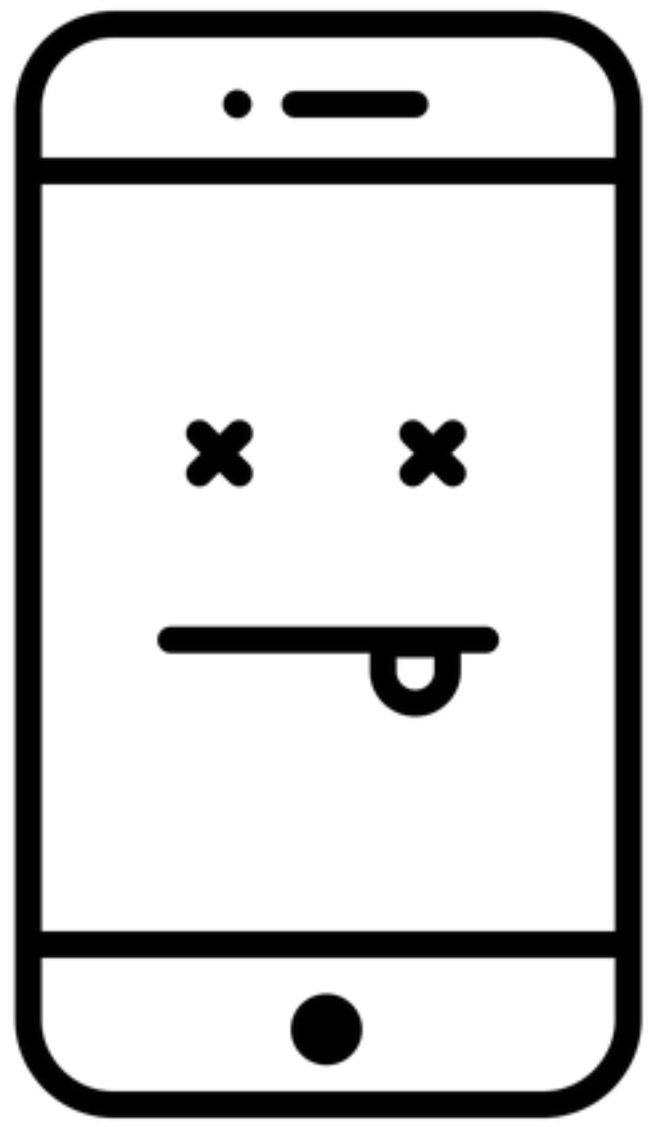
The Resurgence of SQL

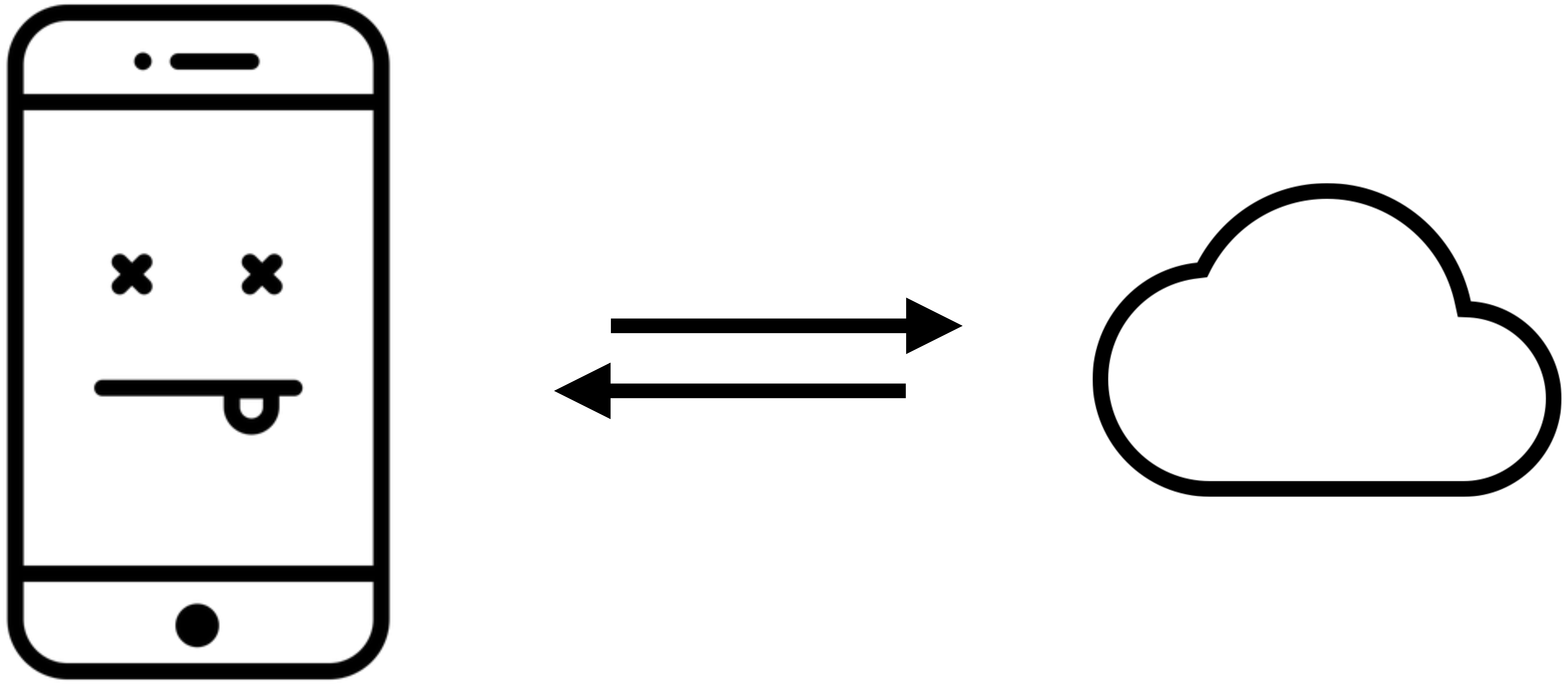
Alec Strong & Jake Wharton

Why Persistence?

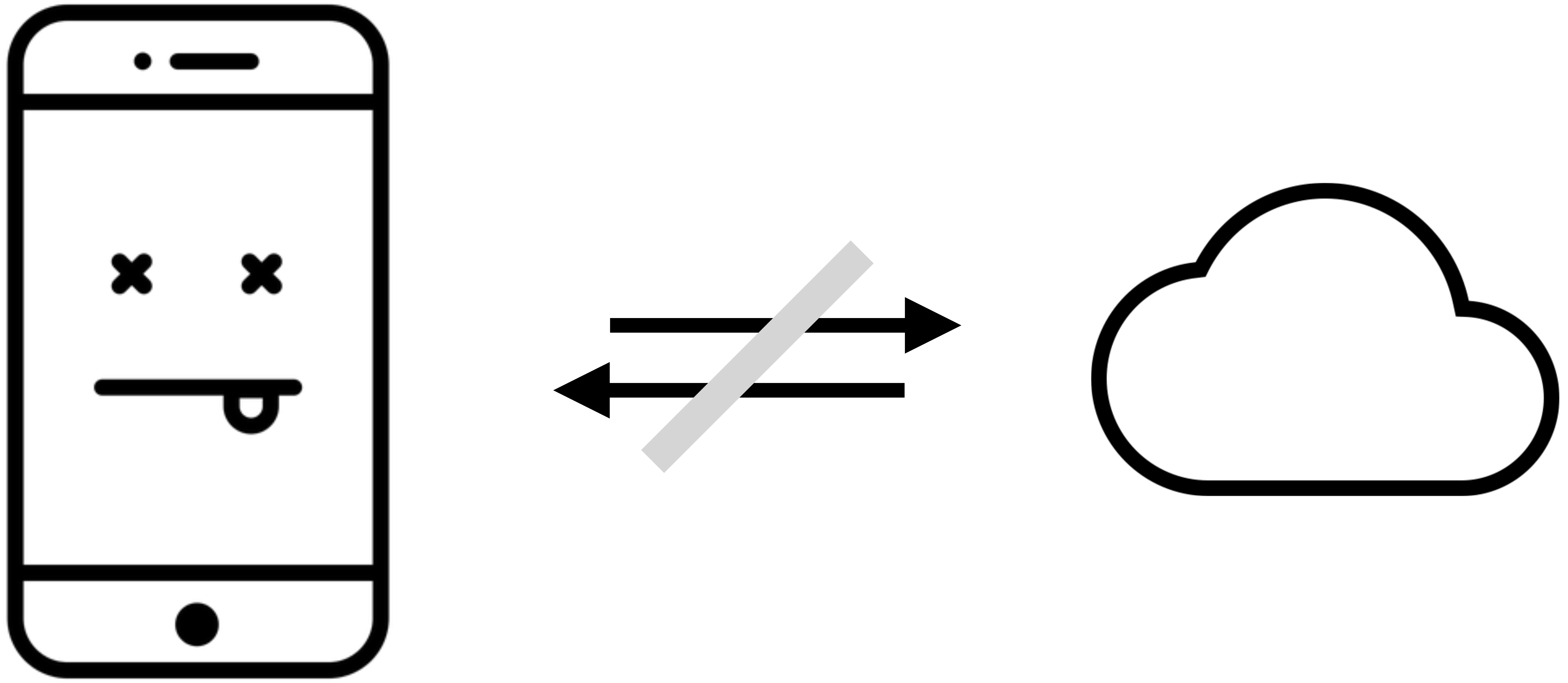
Why Persistence?



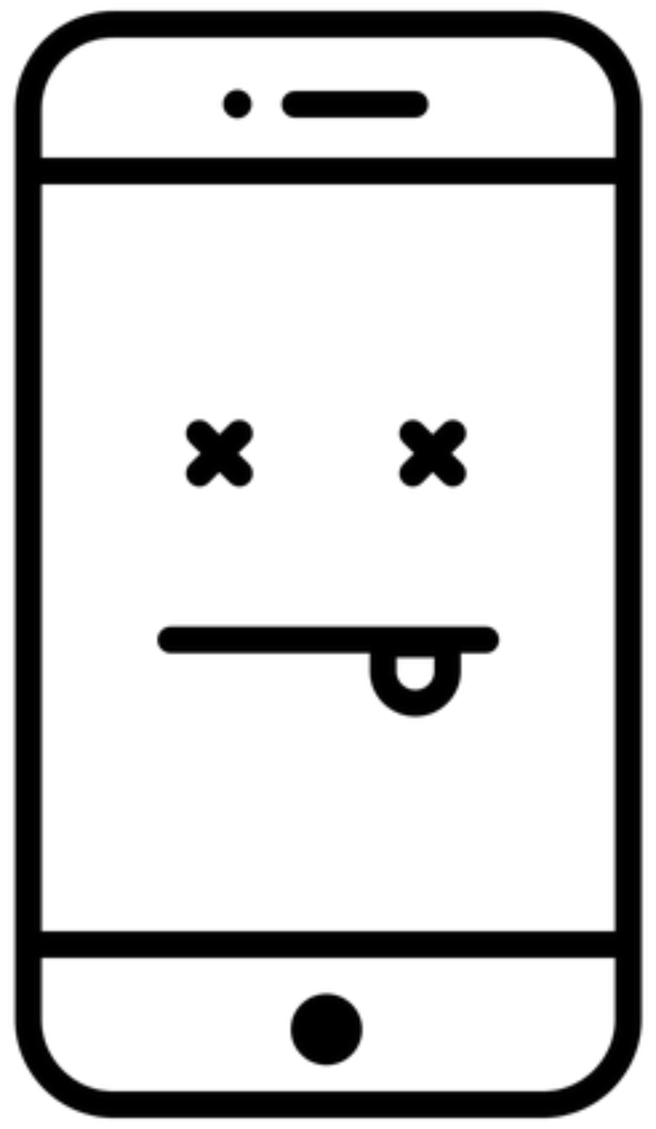
Why Persistence?



Why Persistence?



Why Persistence?



Flat Files

Flat Files

```
sharedPreferences.edit()
    .putString("title", "Shrek")
    .putInt("year", 2001)
    .putFloat("rating", 10.0f)
    .apply()
```

Flat Files

```
sharedPreferences.edit()
    .putString("title", "Shrek")
    .putInt("year", 2001)
    .putFloat("rating", 10.0f)
    .apply()
```

```
<map>
  <string name="title">Shrek</string>
  <int name="year" value="2001" />
  <float name="rating" value="10.0" />
</map>
```

Flat Files

```
sharedPreferences.edit()
    .putString("title", "Shrek")
    .putInt("year", 2001)
    .putFloat("rating", 10.0f)
    .apply()

sharedPreferences.edit()
    .putInt("volume", 8)
    .apply()
```

```
<map>
  <string name="title">Shrek</string>
  <int name="year" value="2001" />
  <float name="rating" value="10.0" />
</map>
```

Flat Files

```
sharedPreferences.edit()
    .putString("title", "Shrek")
    .putInt("year", 2001)
    .putFloat("rating", 10.0f)
    .apply()

sharedPreferences.edit()
    .putInt("volume", 8)
    .apply()
```

```
<map>
  <string name="title">Shrek</string>
  <int name="year" value="2001" />
  <float name="rating" value="10.0" />
  <int name="volume" value="8" />
</map>
```

Flat Files

```
sharedPreferences.edit()
    .putString("title", "Shrek")
    .putInt("year", 2001)
    .putFloat("rating", 10.0f)
    .apply()

sharedPreferences.edit()
    .putInt("volume", 8)
    .apply()
```

```
<map>
  <string name="title">Shrek</string>
  <int name="year" value="2001" />
  <float name="rating" value="10.0" />
  <int name="volume" value="8" />
</map>
```

Flat Files

```kotlin
val sharedPreferences =
    context.getSharedPreferences("user123", MODE_PRIVATE)

sharedPreferences.edit()
    .putString("title", "Shrek")
    .putInt("year", 2001)
    .putFloat("rating", 10.0f)
    .apply()

sharedPreferences.edit()
    .putInt("volume", 8)
    .apply()
```

```xml
<map>
  <string name="title">Shrek</string>
  <int name="year" value="2001" />
  <float name="rating" value="10.0" />
  <int name="volume" value="8" />
</map>
```

Flat Files

```kotlin
val sharedPreferences =
    context.getSharedPreferences("user123", MODE_PRIVATE)

sharedPreferences.edit()
    .putString("title", "Shrek")
    .putInt("year", 2001)
    .putFloat("rating", 10.0f)
    .apply()

sharedPreferences.edit()
    .putInt("volume", 8)
    .apply()
```

```xml
<map>
  <string name="title">Shrek</string>
  <int name="year" value="2001" />
  <float name="rating" value="10.0" />
  <int name="volume" value="8" />
</map>
```

Flat Files

```kotlin
val sharedPreferences =
    context.getSharedPreferences("user123", MODE_PRIVATE)

sharedPreferences.edit()
    .putString("title", "Shrek")
    .putInt("year", 2001)
    .putFloat("rating", 10.0f)
    .apply()

sharedPreferences.edit()
    .putInt("volume", 8)
    .apply()
```

```xml
<map>
  <string name="title">Shrek</string>
  <int name="year" value="2001" />
  <float name="rating" value="10.0" />
  <int name="volume" value="8" />
</map>
```

Flat Files

```
data class User(
    val name: String,
    val age: Int,
    val email: String
)
```

Flat Files

```
data class User(
    val name: String,
    val age: Int,
    val email: String
)

val bob = User("Bob", 20, "bob@bob.bob")
buffer(sink(file)).use {
  adapter.toJson(it, bob)
}
```

Flat Files

```
data class User(
    val name: String,
    val age: Int,
    val email: String
)

val bob = User("Bob", 20, "bob@bob.bob")
buffer(sink(file)).use {
  adapter.toJson(it, bob)
}

{"name":"Bob","age":20,"email":"bob@bob.bob"}
```

Flat Files

```
data class User(
    val name: String,
    val age: Int,
    val email: String
)

val bob = User("Bob", 20, "bob@bob.bob")
buffer(sink(file)).use {
  adapter.toJson(it, bob)
}

{"name":"Bob","age":20,"email":"bob@bob.bob"}
```

Flat Files

```
data class User(
    val name: String,
    val age: Int,
    val email: String,
    val friends: List<User> = emptyList()
)

val alice = User("Alice", 20, "alice@alice.alice")
val bob = User("Bob", 20, "bob@bob.bob", alice)
buffer(sink(file)).use {
  adapter.toJson(it, bob)
}
```

```
{"name":"Bob","age":20,"email":"bob@bob.bob","friends":[{"name":"Alice",
"age":20,"email":"alice@alice.alice"}]}
```

Flat Files

```
data class User(
    val name: String,
    val age: Int,
    val email: String,
    val friends: List<User> = emptyList()
)

val alice = User("Alice", 20, "alice@alice.alice")
val bob = User("Bob", 20, "bob@bob.bob", alice)
buffer(sink(file)).use {
  adapter.toJson(it, bob)
}

{“name":"Bob","age":20,"email":"bob@bob.bob","friends":[{"name":"Alice",
“age":20,"email":"alice@alice.alice"}]}
```

Flat Files

```
data class User(
    val name: String,
    val age: Int,
    val email: String,
    val friends: List<User> = emptyList()
)

val alice = User("Alice", 20, "alice@alice.alice")
val bob = User("Bob", 20, "bob@bob.bob", listOf(alice))
buffer(sink(file)).use {
  adapter.toJson(it, bob)
}
```

```
{“name":"Bob","age":20,"email":"bob@bob.bob","friends":[{"name":"Alice",
“age":20,"email":"alice@alice.alice"}]}
```

Flat Files

```
data class User(
    val name: String,
    val age: Int,
    val email: String,
    val friends: List<User> = emptyList()
)

val alice = User("Alice", 20, "alice@alice.alice")
val bob = User("Bob", 20, "bob@bob.bob", listOf(alice))
buffer(sink(file)).use {
  adapter.toJson(it, bob)
}
```

```
{“name”:”Bob","age":20,"email":"bob@bob.bob","friends":[{"name":"Alice",
“age":20,"email":"alice@alice.alice"}]}
```

Flat Files

```kotlin
data class User(
    val name: String,
    val age: Int,
    val email: String,
    val friends: List<User> = emptyList()
)

val alice = User("Alice", 20, "alice@alice.alice")
val bob = User("Bob", 20, "bob@bob.bob", listOf(alice))
buffer(sink(file)).use {
  adapter.toJson(it, bob)
}

{“name”:”Bob","age":20,"email":"bob@bob.bob","friends":[{"name":"Alice",
“age":20,"email":"alice@alice.alice"}]}
```

Object DBs

```kotlin
data class User(
    val name: String,
    val age: Int,
    val email: String,
    val friends: List<User> = emptyList()
)
```

Object DBs

```
data class User(
    val name: String,
    val age: Int,
    val email: String,
    val friends: List<User> = emptyList()
) : MagicObject()
```

Object DBs

```
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()
```

Object DBs

```kotlin
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()
```

Object DBs

```
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()
```

```
.observeOn(mainThread())
```

Object DBs

```
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()
```

~~.observeOn(mainThread())~~

Object DBs

```
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()

// Users who are 20 or older:
db.where(User::class.java).greaterThanEqualTo("age", 20).findList()
```

Object DBs

```
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()

// Users who are 20 or older:
db.where(User::class.java).greaterThanEqualTo("age", 20).findList()
```

Object DBs

```kotlin
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()

// Users who are 20 or older:
db.where(User::class.java).greaterThanEqualTo("age", 20).findList()
```

Object DBs

```
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()

// Users who are 20 or older:
db.where(User::class.java).greaterThanEqualTo("age", 20).findList()

// Users with 3 or more friends:
// Query all users, count and filter in code :(
```

Object DBs

```kotlin
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()

// Users who are 20 or older:
db.where(User::class.java).greaterThanEqualTo("age", 20).findList()

// Users with 3 or more friends:
// Query all users, count and filter in code :(

// Users friends by Bob (transitively)
// Query friends, friends of friends and combine in code :(
```

Object DBs

```
data class User(
    var name: String,
    var age: Int,
    var email: String,
    var friends: List<User> = emptyList()
) : MagicObject()
```

ORMs

```
data class User(
    val id: Long,
    val name: String,
    val friends: Set<User>
)

data class Checkin(
    val location: String,
    val time: OffsetDateTime,
    val users: Set<User>
)
```

ORMs

```kotlin
@Entity
data class User(
    @Id @GeneratedValue(strategy = AUTO)
    val id: Long,
    val name: String,
    @ManyToMany
    val friends: Set<User>
)

@Entity
data class Checkin(
    val location: String,
    val time: OffsetDateTime,
    @ManyToMany
    val users: Set<User>
)
```

ORMs

```kotlin
@Entity
data class User(
    @Id @GeneratedValue(strategy = AUTO)
    val id: Long,
    val name: String,
    @ManyToMany
    val friends: Set<User>
)

@Entity
data class Checkin(
    val location: String,
    val time: OffsetDateTime,
    @ManyToMany
    val users: Set<User>
)
```

ORMs

```
// Find your friend’s checkins
val me = session.createCritera(User::class.java)
    .add(eq("id", MY_ID)).list().first()
val checkins = session.createCritera(Checkin::class.java)
    .add(eq("users.name", me.friends))
```

ORMs

```
// Find your friend’s checkins
val me = session.createCritera(User::class.java)
    .add(eq("id", MY_ID)).list().first()
val checkins = session.createCritera(Checkin::class.java)
    .add(eq("users.name", me.friends))
```

ORMs

```
@Entity
data class User(
    @Id @GeneratedValue(strategy = AUTO)
    val id: Long,
    val name: String,
    @ManyToMany
    val friends: Set<User>
)

// Find your friend’s checkins
val me = session.createCritera(User::class.java)
    .add(eq("id", MY_ID)).list().first()
val checkins = session.createCritera(Checkin::class.java)
    .add(eq("users.name", me.friends))
```

SQL

- Data Definition Language (DDL)

SQL

- Data Definition Language (DDL)
- Data Manipulation Language (DML)

SQL

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language (DCL)
    - (Not a thing in SQLite)

Data Definition

```sql
CREATE TABLE user (
  _id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL
);

CREATE TABLE friendship (
  friend1 INTEGER NOT NULL REFERENCES user,
  friend2 INTEGER NOT NULL REFERENCES user,
  became_friends INTEGER NOT NULL DEFAULT CURRENT_TIME,
  PRIMARY KEY (friend1, friend2)
);
```

Data Definition

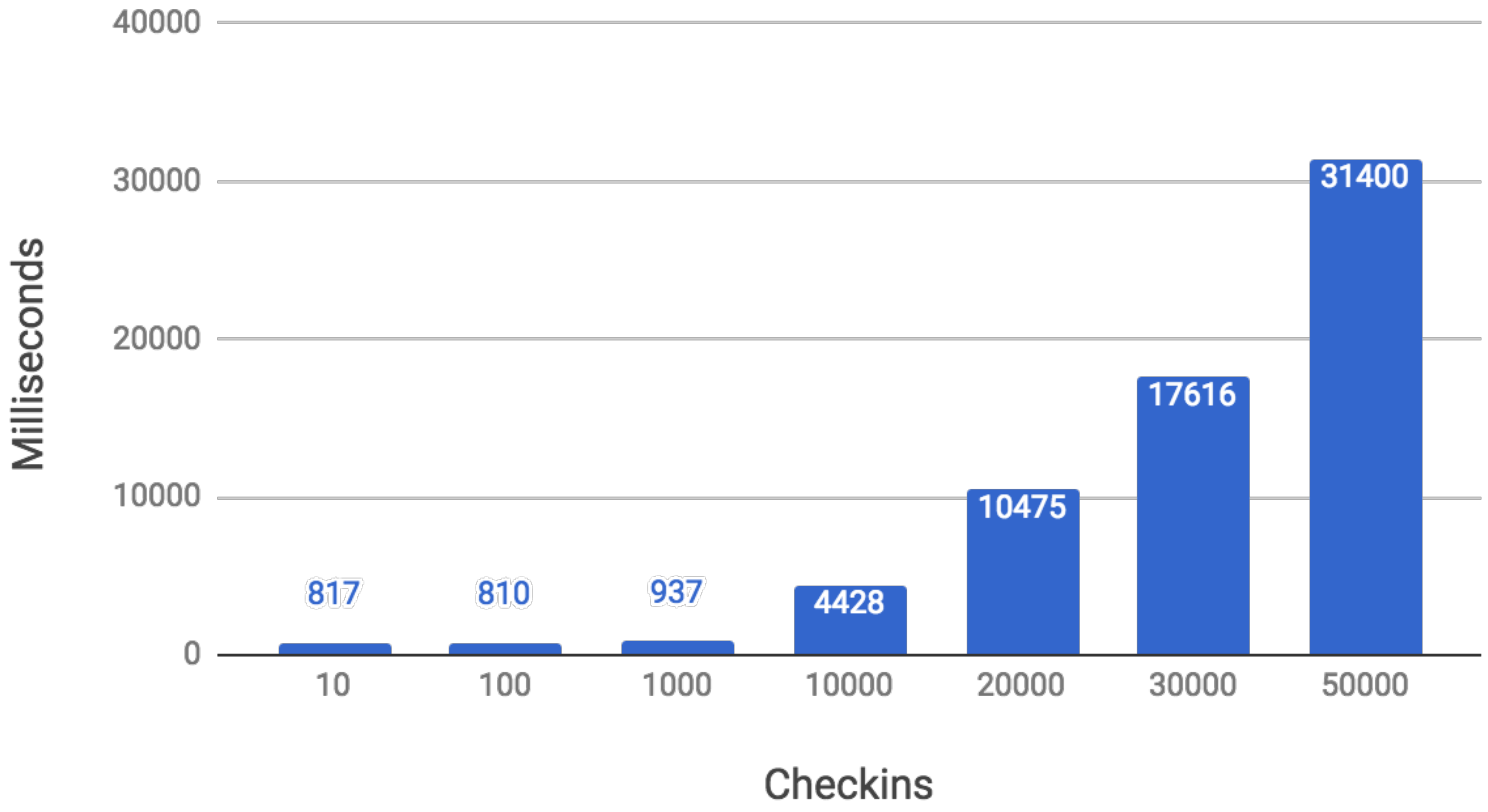
```
CREATE TABLE checkin (
  _id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  time INTEGER NOT NULL
);

CREATE TABLE user_checkin (
  checkin_id INTEGER NOT NULL REFERENCES checkin,
  user_id INTEGER NOT NULL REFERENCES user,
  PRIMARY KEY(checkin_id, user_id)
);
```

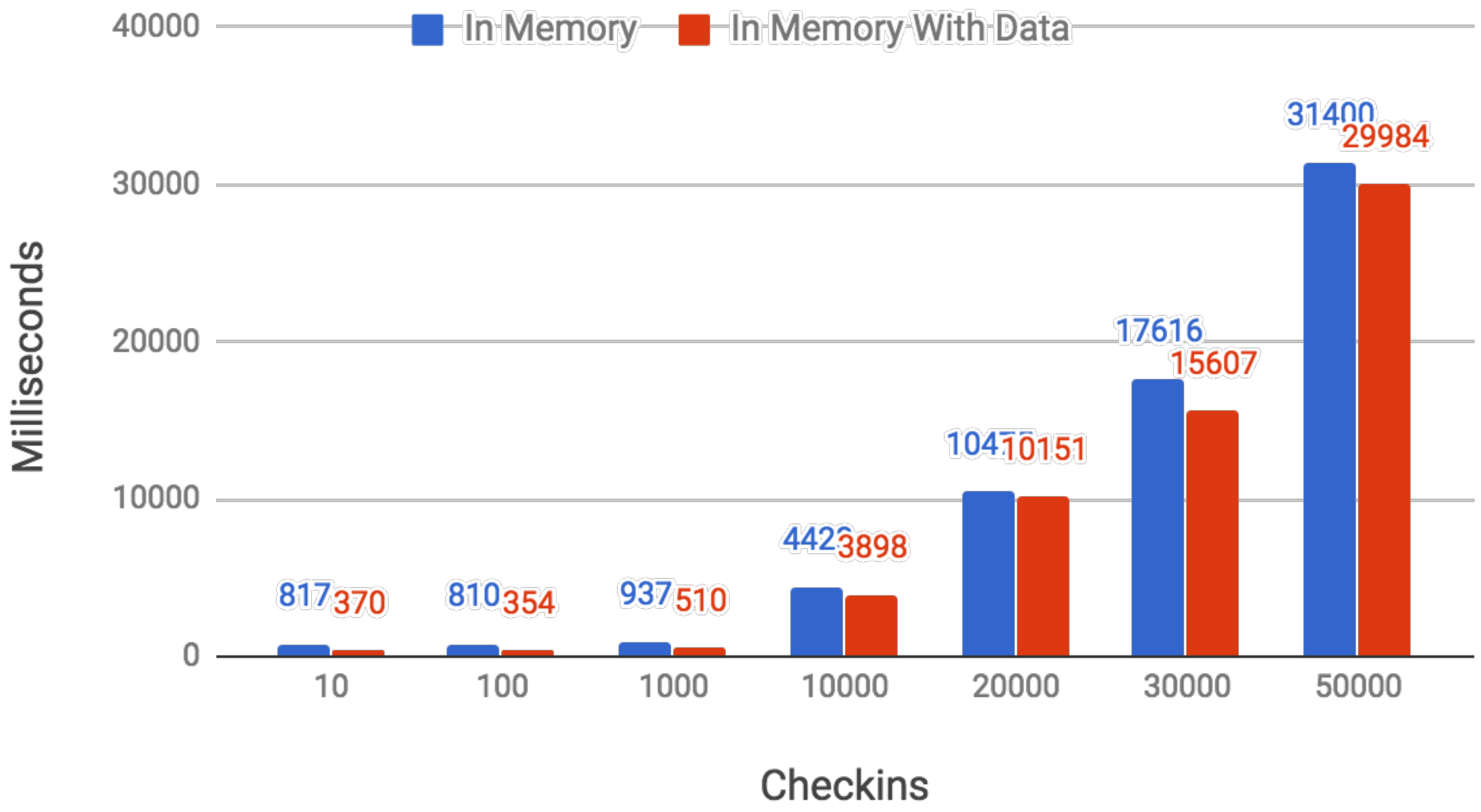
Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

Data Manipulation



Data Manipulation



Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
FROM friendship
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
FROM friendship
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
FROM friendship
WHERE friend1 = MY_ID
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
FROM friendship
WHERE friend1 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
FROM friendship
WHERE friend1 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id


FROM friendship
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id


FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
SELECT friend2
FROM friendship
WHERE friend1 = :my_id

SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id

SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
+
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  var friends = friendships
      .filter { it.friend1 == MY_ID }
      .map { it.friend2 }
  friends += friendships
      .filter { it.friend2 == MY_ID }
      .map { it.friend1 }
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friendships: Collection<Friendship>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

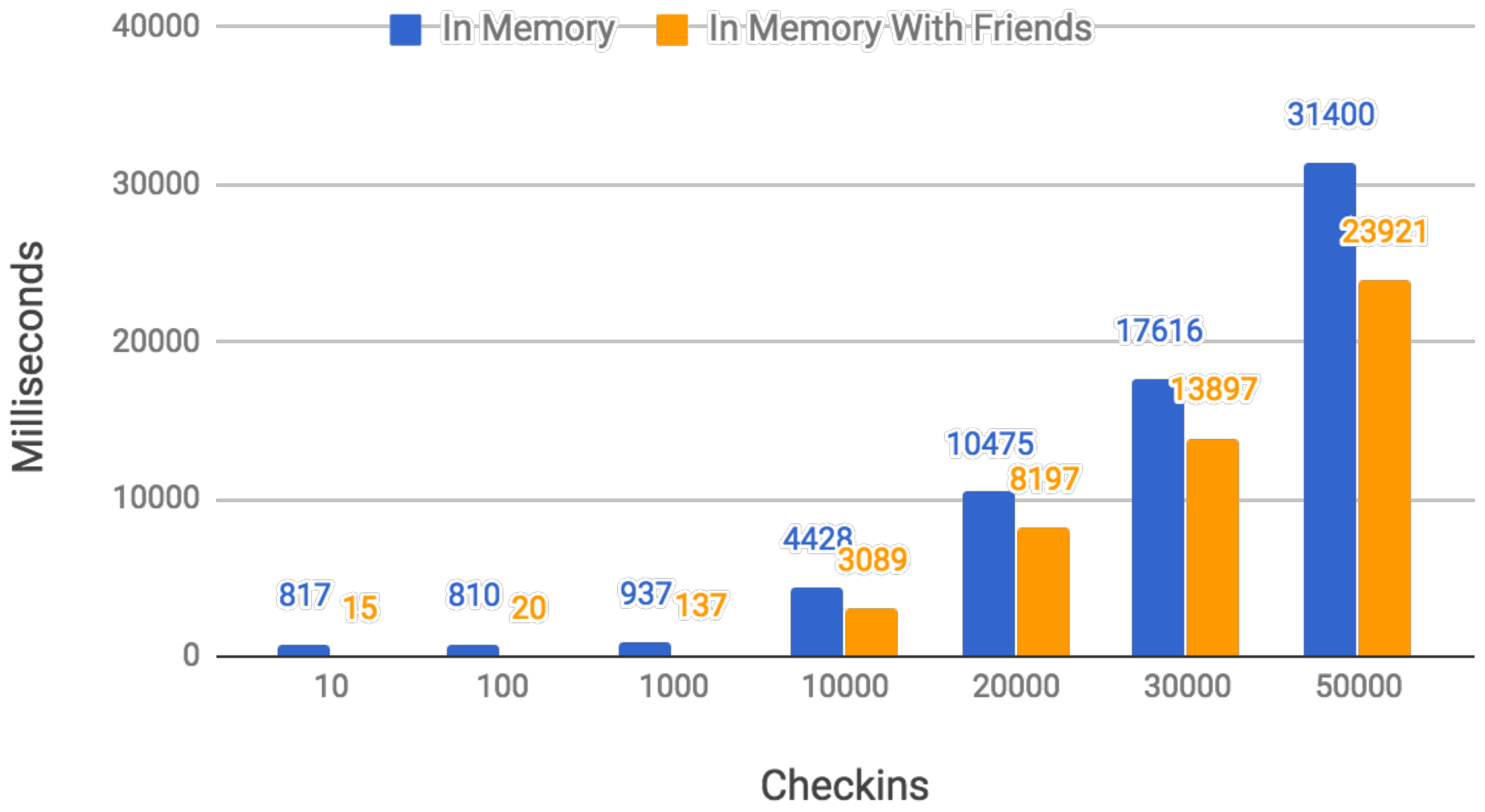
```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation



Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
FROM user_checkin

SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
FROM user_checkin

SELECT friend2
FROM friendship
WHERE friend1 = :my_id
UNION
SELECT friend1
FROM friendship
WHERE friend2 = :my_id
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation

```
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```
SELECT checkin_id
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT checkin_id
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT DISTINCT checkin_id
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT DISTINCT checkin_id
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT
  count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation

```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
  return checkins
      .filter { it.user_id in friends }
      .map { it.checkin_id }
      .distinct()
      .size()
}
```

```sql
SELECT
  count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation

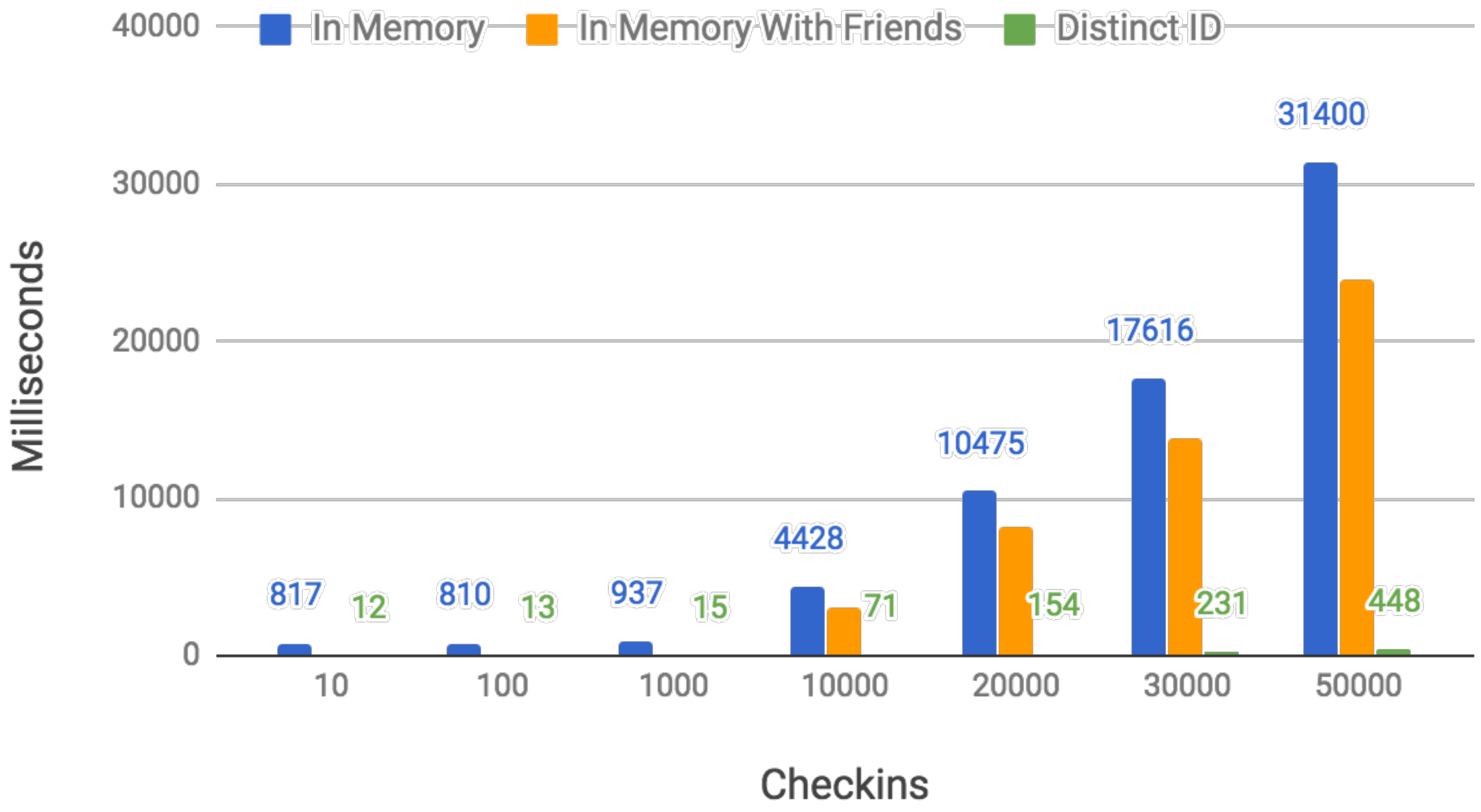
```kotlin
fun friendsCheckins(
    checkins: Collection<UserCheckin>,
    friends: Collection<Long>
): Long {
}
```

```sql
SELECT
  count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

Data Manipulation in SQL!

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

SQL



40000
30000
20000
10000
0
In Memory
In Memory With Friends
Distinct ID
Milliseconds
817
12
810
13
937
15
4428
71
10475
154
17616
231
31400
448
10
100
1000
10000
20000
30000
50000
Checkins

SQL

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

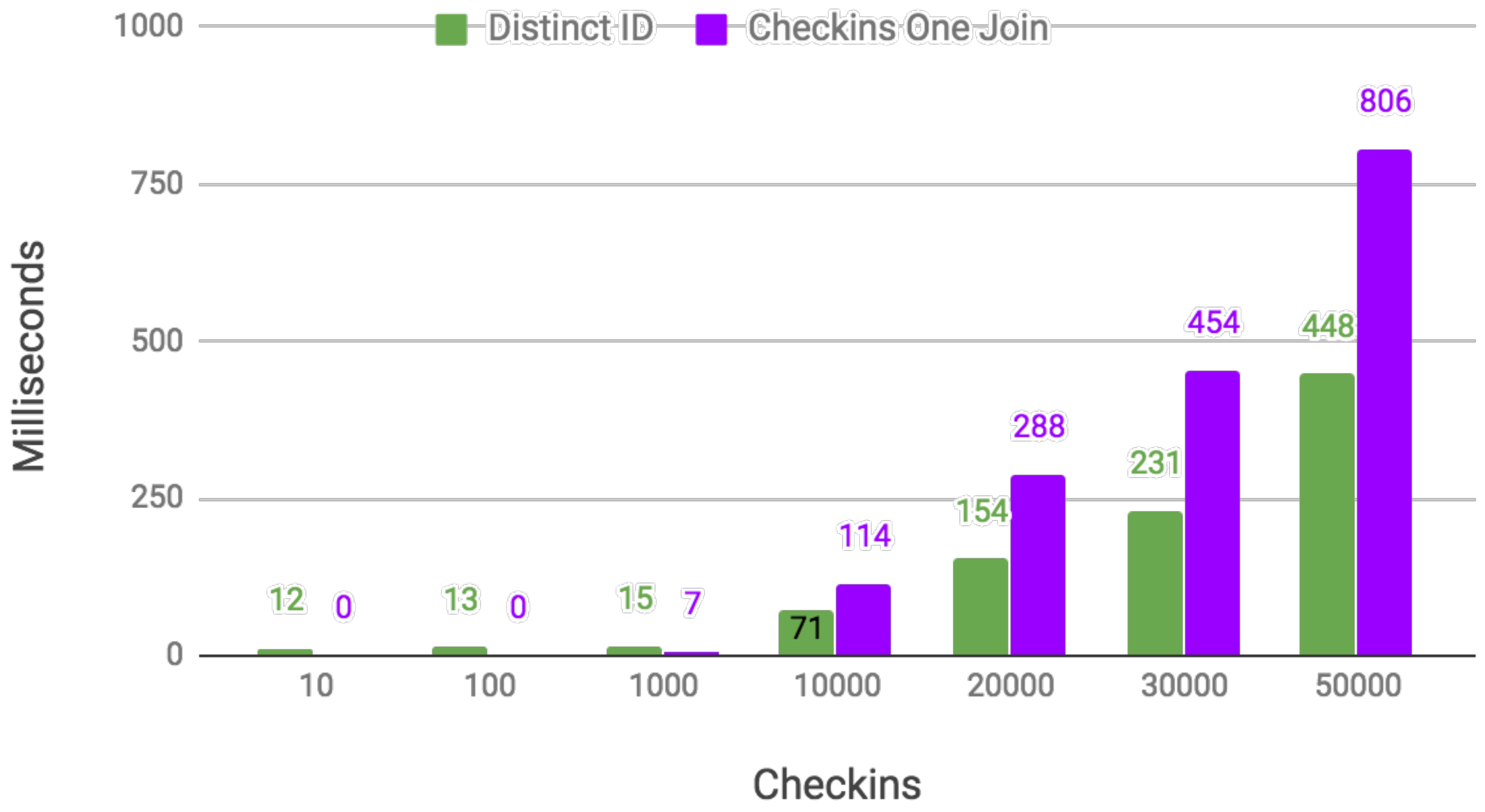
SQL

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

SQL

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

SQL



Distinct ID
Checkins One Join
1000
750
500
250
0
Milliseconds
12
0
13
0
15
7
71
114
154
288
231
454
448
806
10
100
1000
10000
20000
30000
50000
Checkins

Debugging

- Save DB file to external storage and pull with `adb`
  - Stetho

Debugging

- Save DB file to external storage and pull with `adb`
  - Stetho
- View DB file with a SQLite browser (sqlitebrowser)

Debugging

- Save DB file to external storage and pull with `adb`
  - Stetho
- View DB file with a SQLite browser (sqlitebrowser)
- Run queries to learn more

Debugging

- Save DB file to external storage and pull with `adb`
  - Stetho
- View DB file with a SQLite browser (sqlitebrowser)
- Run queries to learn more
- `EXPLAIN QUERY PLAN`

EXPLAIN QUERY PLAN

```
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

EXPLAIN QUERY PLAN

```
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

```sql
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

```
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

Where is this instruction in the `FROM` clause

```
selectid  order  from  detail
"0"       "0"    "0"   "SCAN TABLE user_checkin"
"0"       "1"    "1"   "SEARCH TABLE friendship USING COVERING INDEX
                       sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"       "1"    "1"   "SEARCH TABLE friendship USING COVERING INDEX
                       sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
```

```
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

Full table scan

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

```
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
   (user_id = friend1 AND friend2 = :my_id) OR
   (user_id = friend2 AND friend1 = :my_id)
)
```

Full table scan

Search a subset using an index

selectid	order	from	detail
"0			"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

```
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

The nesting order of this instruction

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

```sql
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

```
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

```
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

```sql
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

```
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"
"0"	"1"	"1"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=? AND friend2=?)"

EXPLAIN QUERY PLAN

```sql
EXPLAIN QUERY PLAN
SELECT count(DISTINCT checkin_id)
FROM user_checkin
JOIN friendship ON (
  (user_id = friend1 AND friend2 = :my_id) OR
  (user_id = friend2 AND friend1 = :my_id)
)
```

EXPLAIN QUERY PLAN

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

```
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
   SELECT friend2
   FROM friendship
   WHERE friend1 = :my_id
   UNION
   SELECT friend1
   FROM friendship
   WHERE friend2 = :my_id
)
```

The subquery ID for the instruction

selectid	order	from	detail
"0"			ckin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	
"0"	"0"	"0"	"SCAN
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

Can either be EXECUTE or EXECUTE CORRELATED

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

EXPLAIN QUERY PLAN

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

EXPLAIN QUERY PLAN

- Subquery is stored, not correlated

EXPLAIN QUERY PLAN

- Subquery is stored, not correlated
- No order of depth, there is only a single scan through the checkin table

EXPLAIN QUERY PLAN

- Subquery is stored, not correlated
- No order of depth, there is only a single scan through the checkin table
- Scans could be searches if we created an index manually

EXPLAIN QUERY PLAN

```
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

```
CREATE INDEX userIdIndex
  ON user_checkin(user_id);
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

```sql
CREATE INDEX userIdIndex
  ON user_checkin(user_id);
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

```sql
CREATE INDEX userIdIndex
  ON user_checkin(user_id);

CREATE INDEX friend2Index
  ON friendship(friend2);
```

selectid	order	from	detail
"0"	"0"	"0"	"SCAN TABLE user_checkin"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SCAN TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

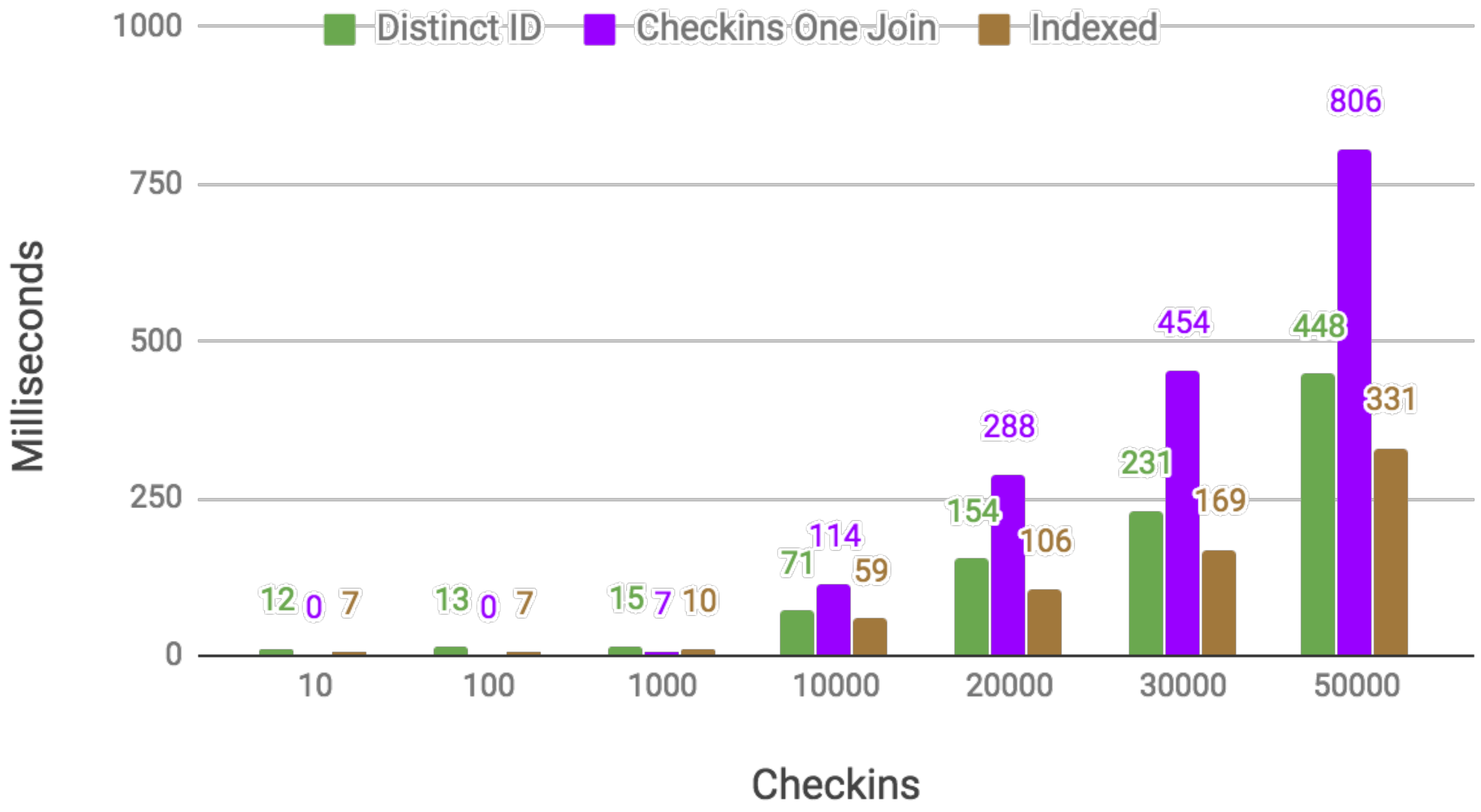
```sql
SELECT count(DISTINCT checkin_id)
FROM user_checkin
WHERE user_id IN (
  SELECT friend2
  FROM friendship
  WHERE friend1 = :my_id
  UNION
  SELECT friend1
  FROM friendship
  WHERE friend2 = :my_id
)
```

```sql
CREATE INDEX userIdIndex
  ON user_checkin(user_id);

CREATE INDEX friend2Index
  ON friendship(friend2);
```

selectid	order	from	detail
"0"	"0"	"0"	"SEARCH TABLE user_checkin USING INDEX userIdIndex (user_id=?)"
"0"	"0"	"0"	"EXECUTE LIST SUBQUERY 1"
"2"	"0"	"0"	"SEARCH TABLE friendship USING COVERING INDEX sqlite_autoindex_friendship_1 (friend1=?)"
"3"	"0"	"0"	"SEARCH TABLE friendship USING INDEX friend2Index (friend2=?)"
"1"	"0"	"0"	"COMPOUND SUBQUERIES 2 AND 3 USING TEMP B-TREE (UNION)"

SQL



Distinct ID
Checkins One Join
Indexed
1000
750
500
250
0
Milliseconds
12
0
7
13
0
7
15
7
10
71
114
59
154
288
106
231
454
169
448
806
331
10
100
1000
10000
20000
30000
50000
Checkins

SQLite and Android

```kotlin
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  companion object {
    private const val VERSION = 1
  }
}
```

SQLite and Android

```
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

SQLite and Android

```kotlin
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

SQLite and Android

```
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE user (
        _id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

SQLite and Android

```
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE user (
        _id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

```
object UserColumns {
  const val TABLE_NAME = "user"

  const val ID = "_id"
  const val NAME = "name"
}
```

SQLite and Android

```kotlin
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE ${UserColumns.TABLE_NAME} (
        ${UserColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${UserColumns.NAME} TEXT NOT NULL
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

```kotlin
object UserColumns {
  const val TABLE_NAME = "user"

  const val ID = "_id"
  const val NAME = "name"
}
```

SQLite and Android

```kotlin
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE ${UserColumns.TABLE_NAME} (
        ${UserColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${UserColumns.NAME} TEXT NOT NULL
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

```kotlin
object UserColumns {
  const val TABLE_NAME = "user"

  const val ID = "_id"
  const val NAME = "name"
}

object FriendshipColumns {
  const val TABLE_NAME = "friendship"

  const val FRIEND_1 = "friend1"
  const val FRIEND_2 = "friend2"
  const val BECAME_FRIENDS = "became_friends"
}
```

SQLite and Android

```kotlin
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE ${UserColumns.TABLE_NAME} (
        ${UserColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${UserColumns.NAME} TEXT NOT NULL
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${FriendshipColumns.TABLE_NAME} (
        ${FriendshipColumns.FRIEND_1} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.FRIEND_2} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.BECAME_FRIENDS} INTEGER NOT NULL DEFAULT CURRENT_TIME,
        PRIMARY KEY (${FriendshipColumns.FRIEND_1}, ${FriendshipColumns.FRIEND_2})
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

```kotlin
object UserColumns {
  const val TABLE_NAME = "user"

  const val ID = "_id"
  const val NAME = "name"
}

object FriendshipColumns {
  const val TABLE_NAME = "friendship"

  const val FRIEND_1 = "friend1"
  const val FRIEND_2 = "friend2"
  const val BECAME_FRIENDS = "became_friends"
}
```

SQLite and Android

```kotlin
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE ${UserColumns.TABLE_NAME} (
        ${UserColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${UserColumns.NAME} TEXT NOT NULL
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${FriendshipColumns.TABLE_NAME} (
        ${FriendshipColumns.FRIEND_1} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.FRIEND_2} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.BECAME_FRIENDS} INTEGER NOT NULL DEFAULT CURRENT_TIME,
        PRIMARY KEY (${FriendshipColumns.FRIEND_1}, ${FriendshipColumns.FRIEND_2})
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

```kotlin
object UserColumns {
  const val TABLE_NAME = "user"

  const val ID = "_id"
  const val NAME = "name"
}

object FriendshipColumns {
  const val TABLE_NAME = "friendship"

  const val FRIEND_1 = "friend1"
  const val FRIEND_2 = "friend2"
  const val BECAME_FRIENDS = "became_friends"
}

object CheckinColumns {
  const val TABLE_NAME = "checkin"

  const val ID = "_id"
  const val NAME = "name"
  const val TIME = "time"
}
```

SQLite and Android

```
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE ${UserColumns.TABLE_NAME} (
        ${UserColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${UserColumns.NAME} TEXT NOT NULL
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${FriendshipColumns.TABLE_NAME} (
        ${FriendshipColumns.FRIEND_1} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.FRIEND_2} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.BECAME_FRIENDS} INTEGER NOT NULL DEFAULT CURRENT_TIME,
        PRIMARY KEY (${FriendshipColumns.FRIEND_1}, ${FriendshipColumns.FRIEND_2})
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${CheckinColumns.TABLE_NAME} (
        ${CheckinColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${CheckinColumns.NAME} TEXT NOT NULL,
        ${CheckinColumns.TIME} INTEGER NOT NULL
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

```
object UserColumns {
  const val TABLE_NAME = "user"

  const val ID = "_id"
  const val NAME = "name"
}

object FriendshipColumns {
  const val TABLE_NAME = "friendship"

  const val FRIEND_1 = "friend1"
  const val FRIEND_2 = "friend2"
  const val BECAME_FRIENDS = "became_friends"
}

object CheckinColumns {
  const val TABLE_NAME = "checkin"

  const val ID = "_id"
  const val NAME = "name"
  const val TIME = "time"
}
```

SQLite and Android

```kotlin
class MyDatabase(context: Context, name: String?)
    : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE ${UserColumns.TABLE_NAME} (
        ${UserColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${UserColumns.NAME} TEXT NOT NULL
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${FriendshipColumns.TABLE_NAME} (
        ${FriendshipColumns.FRIEND_1} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.FRIEND_2} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.BECAME_FRIENDS} INTEGER NOT NULL DEFAULT CURRENT_TIME,
        PRIMARY KEY (${FriendshipColumns.FRIEND_1}, ${FriendshipColumns.FRIEND_2})
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${CheckinColumns.TABLE_NAME} (
        ${CheckinColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${CheckinColumns.NAME} TEXT NOT NULL,
        ${CheckinColumns.TIME} INTEGER NOT NULL
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

```kotlin
object UserColumns {
  const val TABLE_NAME = "user"

  const val ID = "_id"
  const val NAME = "name"
}

object FriendshipColumns {
  const val TABLE_NAME = "friendship"

  const val FRIEND_1 = "friend1"
  const val FRIEND_2 = "friend2"
  const val BECAME_FRIENDS = "became_friends"
}

object CheckinColumns {
  const val TABLE_NAME = "checkin"

  const val ID = "_id"
  const val NAME = "name"
  const val TIME = "time"
}

object UserCheckinColumns {
  const val TABLE_NAME = "user_checkin"

  const val CHECKIN_ID = "checkin_id"
  const val USER_ID = "user_id"
}
```

SQLite and Android

```
class MyDatabase(context: Context, name: String?) : SQLiteOpenHelper(context, name, null, VERSION) {
  override fun onCreate(db: SQLiteDatabase) {
    db.execSQL("""
      CREATE TABLE ${UserColumns.TABLE_NAME} (
        ${UserColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${UserColumns.NAME} TEXT NOT NULL
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${FriendshipColumns.TABLE_NAME} (
        ${FriendshipColumns.FRIEND_1} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.FRIEND_2} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        ${FriendshipColumns.BECAME_FRIENDS} INTEGER NOT NULL DEFAULT CURRENT_TIME,
        PRIMARY KEY (${FriendshipColumns.FRIEND_1}, ${FriendshipColumns.FRIEND_2})
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${CheckinColumns.TABLE_NAME} (
        ${CheckinColumns.ID} INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
        ${CheckinColumns.NAME} TEXT NOT NULL,
        ${CheckinColumns.TIME} INTEGER NOT NULL
      );
    """.trimIndent())

    db.execSQL("""
      CREATE TABLE ${UserCheckinColumns.TABLE_NAME} (
        ${UserCheckinColumns.CHECKIN_ID} INTEGER NOT NULL REFERENCES ${CheckinColumns.TABLE_NAME},
        ${UserCheckinColumns.USER_ID} INTEGER NOT NULL REFERENCES ${UserColumns.TABLE_NAME},
        PRIMARY KEY(${UserCheckinColumns.CHECKIN_ID}, ${UserCheckinColumns.USER_ID})
      );
    """.trimIndent())
  }

  override fun onUpgrade(db: SQLiteDatabase, old: Int, new: Int) {
  }

  companion object {
    private const val VERSION = 1
  }
}
```

```
object UserColumns {
  const val TABLE_NAME = "user"

  const val ID = "_id"
  const val NAME = "name"
}

object FriendshipColumns {
  const val TABLE_NAME = "friendship"

  const val FRIEND_1 = "friend1"
  const val FRIEND_2 = "friend2"
  const val BECAME_FRIENDS = "became_friends"
}

object CheckinColumns {
  const val TABLE_NAME = "checkin"

  const val ID = "_id"
  const val NAME = "name"
  const val TIME = "time"
}

object UserCheckinColumns {
  const val TABLE_NAME = "user_checkin"

  const val CHECKIN_ID = "checkin_id"
  const val USER_ID = "user_id"
}
```

SQLite and Android

```
fun friendsCheckins(db: SQLiteDatabase, myId: Int): Cursor? {
  return db.rawQuery("""
    SELECT count(DISTINCT ${UserCheckinColumns.CHECKIN_ID})
    FROM ${UserCheckinColumns.TABLE_NAME}
    WHERE ${UserCheckinColumns.USER_ID} IN (
      SELECT ${FriendshipColumns.FRIEND_2}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_1} = ?1
      UNION
      SELECT ${FriendshipColumns.FRIEND_1}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_2} = ?1
    )
  """.trimIndent(), arrayOf(myId.toString()))
}
```

SQLite and Android

```
fun friendsCheckins(db: SQLiteDatabase, myId: Int): Cursor? {
  return db.rawQuery("""
    SELECT count(DISTINCT ${UserCheckinColumns.CHECKIN_ID})
    FROM ${UserCheckinColumns.TABLE_NAME}
    WHERE ${UserCheckinColumns.USER_ID} IN (
      SELECT ${FriendshipColumns.FRIEND_2}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_1} = ?1
      UNION
      SELECT ${FriendshipColumns.FRIEND_1}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_2} = ?1
    )
  """.trimIndent(), arrayOf(myId.toString()))
}
```

SQLite and Android

```
fun friendsCheckins(db: SQLiteDatabase, myId: Int): Cursor? {
  return db.rawQuery("""
    SELECT count(DISTINCT ${UserCheckinColumns.CHECKIN_ID})
    FROM ${UserCheckinColumns.TABLE_NAME}
    WHERE ${UserCheckinColumns.USER_ID} IN (
      SELECT ${FriendshipColumns.FRIEND_2}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_1} = ?1
      UNION
      SELECT ${FriendshipColumns.FRIEND_1}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_2} = ?1
    )
  """.trimIndent(), arrayOf(myId.toString()))
}
```

SQLite and Android

```
fun friendsCheckins(db: SQLiteDatabase, myId: Int): Int {
  return db.rawQuery("""
    SELECT count(DISTINCT ${UserCheckinColumns.CHECKIN_ID})
    FROM ${UserCheckinColumns.TABLE_NAME}
    WHERE ${UserCheckinColumns.USER_ID} IN (
      SELECT ${FriendshipColumns.FRIEND_2}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_1} = ?1
      UNION
      SELECT ${FriendshipColumns.FRIEND_1}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_2} = ?1
    )
  """.trimIndent(), arrayOf(myId.toString()))
}
```

SQLite and Android

```
fun friendsCheckins(db: SQLiteDatabase, myId: Int): Int {
  val cursor = db.rawQuery("""
    SELECT count(DISTINCT ${UserCheckinColumns.CHECKIN_ID})
    FROM ${UserCheckinColumns.TABLE_NAME}
    WHERE ${UserCheckinColumns.USER_ID} IN (
      SELECT ${FriendshipColumns.FRIEND_2}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_1} = ?1
      UNION
      SELECT ${FriendshipColumns.FRIEND_1}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_2} = ?1
    )
  """.trimIndent(), arrayOf(myId.toString()))
  cursor.use {
    if (it.moveToNext()) {
      return it.getInt(0)
    }
    throw IllegalStateException("Query returned zero rows")
  }
}
```

SQLite and Android

```
fun friendsCheckins(db: SQLiteDatabase, myId: Int): Int {
  val count = "checkin_count"
  val cursor = db.rawQuery("""
    SELECT count(DISTINCT ${UserCheckinColumns.CHECKIN_ID}) AS $count
    FROM ${UserCheckinColumns.TABLE_NAME}
    WHERE ${UserCheckinColumns.USER_ID} IN (
      SELECT ${FriendshipColumns.FRIEND_2}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_1} = ?1
      UNION
      SELECT ${FriendshipColumns.FRIEND_1}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_2} = ?1
    )
  """.trimIndent(), arrayOf(myId.toString()))
  cursor.use {
    if (it.moveToNext()) {
      return it.getInt(it.getColumnIndex(count))
    }
    throw IllegalStateException("Query returned zero rows")
  }
}
```

SQLite and Android

```
fun friendsCheckins(db: SQLiteDatabase, myId: Int): Int {
  val count = "checkin_count"
  val cursor = db.rawQuery("""
    SELECT count(DISTINCT ${UserCheckinColumns.CHECKIN_ID}) AS $count
    FROM ${UserCheckinColumns.TABLE_NAME}
    WHERE ${UserCheckinColumns.USER_ID} IN (
      SELECT ${FriendshipColumns.FRIEND_2}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_1} = ?1
      UNION
      SELECT ${FriendshipColumns.FRIEND_1}
      FROM ${FriendshipColumns.TABLE_NAME}
      WHERE ${FriendshipColumns.FRIEND_2} = ?1
    )
  """.trimIndent(), arrayOf(myId.toString()))
  cursor.use {
    if (it.moveToNext()) {
      return it.getInt(it.getColumnIndex(count))
    }
    throw IllegalStateException("Query returned zero rows")
  }
}
```

SQLite and Android

- Strings... Strings everywhere...

SQLite and Android

- Strings... Strings everywhere...
- No query or type safety

SQLite and Android

- Strings... Strings everywhere...
- No query or type safety
- Prohibitive level of verbosity

SQLite and Android

- Strings... Strings everywhere...
- No query or type safety
- Prohibitive level of verbosity
- Reactive updates only achievable through a `ContentProvider`
  - Another level of hell, omitted for sanity

SQLDelight & Room

SQLDelight

Room

SQLDelight

```
CREATE TABLE todo_list (
  _id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  archived INTEGER AS Boolean NOT NULL DEFAULT 0
);
```

```
@AutoValue
public abstract class TodoList implements Parcelable, TodoListModel {
  public static final Factory<TodoList> FACTORY =
      new TodoListModel.Factory<>(AutoValue_TodoList::new);
}
```

Room

```
@Entity(tableName = "todo_list")
data class TodoList(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "_id")
    val id: Int = 0,
    val name: String,
    val archived: Boolean = false
)
```

SQLDelight

- No restriction on Java or SQL type

Room

SQLDelight

- No restriction on Java or SQL type

Room

- No restriction on Java type (`@Ignore`)

SQLDelight

- No restriction on Java or SQL type
- No inheritance or other OOP

Room

- No restriction on Java type (`@Ignore`)

SQLDelight

- No restriction on Java or SQL type
- No inheritance or other OOP

Room

- No restriction on Java type (`@Ignore`)
- Subset of SQLite supported
  - `UNIQUE`, `CHECK`, `DEFAULT`

SQLDelight

- No restriction on Java or SQL type
- No inheritance or other OOP
- Doesn't play well with Kotlin `data` classes

Room

- No restriction on Java type (`@Ignore`)
- Subset of SQLite supported
  - `UNIQUE`, `CHECK`, `DEFAULT`

SQLDelight

- No restriction on Java or SQL type
- No inheritance or other OOP
- Doesn't play well with Kotlin `data` classes

Room

- No restriction on Java type (`@Ignore`)
- Subset of SQLite supported
  - `UNIQUE`, `CHECK`, `DEFAULT`
- Doesn't work with AutoValue

SQLDelight

```
CREATE TABLE todo_item (
  _id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  todo_list_id INTEGER NOT NULL REFERENCES todo_list,
  description TEXT NOT NULL,
  complete INTEGER AS Boolean NOT NULL DEFAULT 0
);

createListIdIndex:
CREATE INDEX item_list_id ON todo_item(todo_list_id);


@AutoValue
public abstract class TodoItem implements TodoItemModel, Parcelable {
  public static final Factory<TodoItem> FACTORY =
      new Factory<>(AutoValue_TodoItem::new);
}
```

Room

```kotlin
@Entity(
    tableName = "todo_item",
    foreignKeys = arrayOf(ForeignKey(
        entity = TodoItem::class,
        parentColumns = arrayOf("_id"),
        childColumns = arrayOf("todo_list_id")
    ))
)
data class TodoItem(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "_id")
    val id: Long,
    @ColumnInfo(name = "todo_list_id", index = true)
    val todoListId: Long,
    val description: String,
    val complete: Boolean = false
)
```

SQLDelight

```
insertList:
INSERT INTO todo_list (name)
VALUES (?);
```

```
private val insertList: InsertList by lazy {
  InsertList(db.writableDatabase)
}

...

db.bindAndExecute(insertList) { bind(name) }
```

Room

```
@Insert
fun insert(list: TodoList)

listDao.insert(TodoList(name = name))
```

SQLDelight

- Can’t insert an object

Room

- Can only insert objects

SQLDelight

- Can’t insert an object
- Verbose - requires you maintain the cache of mutator queries

Room

- Can only insert objects

SQLDelight

- Can’t insert an object
- Verbose - requires you maintain the cache of mutator queries

Room

- Can only insert objects
- Can’t use DAO’s during creation

SQLDelight

```
titleAndCount:
SELECT name, count(todo_item._id)
FROM todo_list
LEFT JOIN todo_item ON (todo_list._id = todo_list_id)
WHERE todo_list._id = ? AND complete = 0
GROUP BY todo_list._id;
```

```
@AutoValue
public abstract class TitleAndCount implements TitleAndCountModel {
  public static final TitleAndCountCreator CREATOR
      = AutoValue_TitleAndCount::new;
}
```

Room

```
@Query("" +
    "SELECT name, count(*) AS count\n" +
    "FROM todo_list\n" +
    "LEFT JOIN todo_item ON (todo_list._id = todo_list_id)\n" +
    "WHERE todo_list._id = :todoListId AND complete = 0\n" +
    "GROUP BY todo_list._id"
)
fun titleAndCount(todoListId: Long): Flowable<TitleAndCount>


data class TitleAndCount(
    val name: String,
    val count: Int
)
```

SQLDelight

Room

- “Not sure how to convert a Cursor to this method's return type”

SQLDelight

Room

- “Not sure how to convert a Cursor to this method's return type”
- Not type safe

Not type safe

```
data class TitleAndCount(
    val name: String,
    val count: Int
)


println(name)     // Grocery List
println(count)    // 4
```

Not type safe

```
data class TitleAndCount(
    val name: String,
    val count: Int
)
```

Not type safe

```
data class TitleAndCount(
    val names: String,
    val count: Int
)
```

```
java.lang.IllegalArgumentException: Parameter specified as non-null is null
```

Not type safe

```
data class TitleAndCount(
    val name: String,
    val count: Int
)
```

Not type safe

```
data class TitleAndCount(
    val name: Int,
    val count: Int
)


println(name)     // 0
println(count)    // 4
```

Room

```
itemDao.titleAndCount(listId)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe { titleAndCount ->
      TODO()
    }
```

SQLDelight

```
db.createQuery(TodoItem.FACTORY.titleAndCount(listId))
    .mapToOne(TitleAndCount.MAPPER::map)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe {
      TODO()
    }
```

```
public static final RowMapper<TodoItem> MAPPER =
    TodoItem.FACTORY.titleAndCountMapper(CREATOR);
```

SQLDelight

```
db.createQuery(TodoItem.FACTORY.titleAndCount(listId))
    .mapToOne(TitleAndCount.MAPPER::map)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe {
      TODO()
    }
```

Not type safe

```
db.createQuery(TodoItem.FACTORY.titleAndCount(listId))

    // Wreak havoc

    .mapToOne(TitleAndCount.MAPPER::map)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe {
      TODO()
    }
```

SQLDelight

Room

- “Not sure how to convert a Cursor to this method's return type”
- Not type safe

SQLDelight

- Verbose calling code

Room

- “Not sure how to convert a Cursor to this method's return type”
- Not type safe

SQLDelight

- Verbose calling code
- SQLBrite - SQLDelight bridge not type safe

Room

- “Not sure how to convert a Cursor to this method's return type”
- Not type safe

SQLDelight

- Full DDL/DML support
  - Views, Triggers, Indexes

Room

SQLDelight

- Full DDL/DML support
  - Views, Triggers, Indexes
- Compiler error == IDE error

Room

SQLDelight

- Full DDL/DML support
  - Views, Triggers, Indexes
- Compiler error == IDE error
- Better tooling
  - Find usages, refactoring, auto complete

Room

SQLDelight

- Full DDL/DML support
  - Views, Triggers, Indexes
- Compiler error == IDE error
- Better tooling
  - Find usages, refactoring, auto complete

Room

- Migration testing utilities

SQLDelight

- Full DDL/DML support
  - Views, Triggers, Indexes
- Compiler error == IDE error
- Better tooling
  - Find usages, refactoring, auto complete

Room

- Migration testing utilities
- Embedded object types

SQLDelight

- Full DDL/DML support
  - Views, Triggers, Indexes
- Compiler error == IDE error
- Better tooling
  - Find usages, refactoring, auto complete

Room

- Migration testing utilities
- Embedded object types
- Better support/documentation

SQLDelight

Room

SQLDelight

- Associate a java type to a column definition and receive type safe projections and mutation apis.

Room

- Define a table in java and serialize java objects to and from a query

Room

- Unless you have a reason to otherwise, use Room

Room

- Unless you have a reason to otherwise, use Room
- Support and documentation is way better

Room

- Unless you have a reason to otherwise, use Room
- Support and documentation is way better
- More than enough sqlite support to get the benefits you need

Room

- Unless you have a reason to otherwise, use Room
- Support and documentation is way better
- More than enough sqlite support to get the benefits you need
- API feels familiar and simple — akin to Retrofit

SQLDelight

- Spending a lot of time in SQLite → Better tooling

SQLDelight

- Spending a lot of time in SQLite → Better tooling
- Complicated models → Type Safety

SQLDelight

- Spending a lot of time in SQLite → Better tooling
- Complicated models → Type Safety
- Complicated client backend → Views, Triggers, Virtual Tables, Inserts

SQLDelight 1.0

- `working-kotlin` branch on GitHub

SQLDelight pre-1.0

```sql
CREATE TABLE todo_list (
  _id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  archived INTEGER AS Boolean NOT NULL DEFAULT 0
);
```

```java
@AutoValue
public abstract class TodoList implements Parcelable, TodoListModel {
  public static final Factory<TodoList> FACTORY =
      new TodoListModel.Factory<>(AutoValue_TodoList::new);
}
```

SQLDelight 1.0

```
CREATE TABLE todo_list (
  _id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL,
  archived INTEGER AS Boolean NOT NULL DEFAULT 0
);
```

SQLDelight pre-1.0

```
insertList:
INSERT INTO todo_list (name)
VALUES (?);
```

```
private val insertList: InsertList by lazy {
  InsertList(db.writableDatabase)
}

...

db.bindAndExecute(insertList) { bind(name) }
```

SQLDelight 1.0

```
insertList:
INSERT INTO todo_list (name)
VALUES (?);
```

```
db.insertList(name)
```

SQLDelight pre-1.0

```
db.createQuery(TodoItem.FACTORY.titleAndCount(listId))
    .mapToOne(TitleAndCount.MAPPER::map)
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe {
      TODO()
    }
```

```
public static final RowMapper<TodoItem> MAPPER =
    TodoItem.FACTORY.titleAndCountMapper(CREATOR);
```

SQLDelight 1.0

```
db.titleAndCount(listId).observe()
    .mapToOne()
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe {
      TODO()
    }
```

SQLDelight 1.0

```
db.titleAndCount(::CustomType, listId).observe()
    .mapToOne()
    .observeOn(AndroidSchedulers.mainThread())
    .subscribe {
      TODO()
    }


fun <T> titleAndCount(
    mapper: (title: String, count: Int) -> T,
    listId: Long
): Query<T>
```

SQLDelight 1.0

- `working-kotlin` branch on GitHub

SQLDelight 1.0

- `working-kotlin` branch on GitHub
- Increase precision of observable emissions
  - Only possible because SQLDelight is a compiler

SQLDelight 1.0

- `working-kotlin` branch on GitHub
- Increase precision of observable emissions
  - Only possible because SQLDelight is a compiler
- Embrace Kotlin as the future

Future of SQLite on Android

- SupportSQLite

SupportSQLite

```
SQLiteDatabase
SQLiteOpenHelper
SQLiteProgram
SQLiteStatement
```

SupportSQLite

```
SupportSQLiteDatabase
SupportSQLiteOpenHelper
SupportSQLiteProgram
SupportSQLiteStatement
```

SupportSQLite

```
SupportSQLiteDatabase
SupportSQLiteOpenHelper
SupportSQLiteProgram
SupportSQLiteStatement
SupportSQLiteQuery

SupportSQLiteDatabase.query(supportQuery)
```

SupportSQLite

```
api 'android.arch.persistence:db:1.0.0-beta1'
```

SupportSQLite

```
api 'android.arch.persistence:db:1.0.0-beta1'

implementation 'android.arch.persistence:db-framework:1.0.0-beta1'
```

Future of SQLite on Android

- SupportSQLite
- Paging

Paging

- Enables efficient paging of large data sources

Paging

- Enables efficient paging of large data sources
- Not tied to SQL, Room, or RecyclerView

Paging

- Enables efficient paging of large data sources
- Not tied to SQL, Room, or RecyclerView
- Seamless Room support

```
@Query("select * from users WHERE age > :age order by name DESC")
fun usersOlderThan(age: Int): TiledDataSource<User>
```

```sql
SELECT *
FROM persistence_solution
WHERE type != 'flat'
  AND type != 'ORM'
  AND type != 'ObjectDB'
```

The Resurgence of SQL

@Strongolopolis & @JakeWharton