CSE 162 Mobile Computing Lecture 16: Context-Aware Computing and SQL

Hua Huang

Context Awareness in Mobile Systems

Ubicomp - Physical World Computing

„In the 21st century the technology revolution will move into the everyday, the small and the invisible..."

Mark Weiser (1952 – 1999), XEROX PARC

"Ubiquitous Computing enhances computer use by making computers available throughout the physical environment, while making them effectively invisible to the user"

- Ubicomp: a field on a physical world richly and invisibly interwoven with sensors, actuators, displays, and computational elements, embedded seamlessly in everyday objects of lives and connected through a continuous network.

  - Mark Weiser in his last article in IBM Sys. Journal, 1999

What are contexts

- A context represents the state or situation in the environment of a system that affects that system's (application specific) behaviour

Types of Contexts, a systematic view

- Physical Contexts
  - What
  - Where
  - When
- Computing System Contexts
  - How
- User Context
  - Who

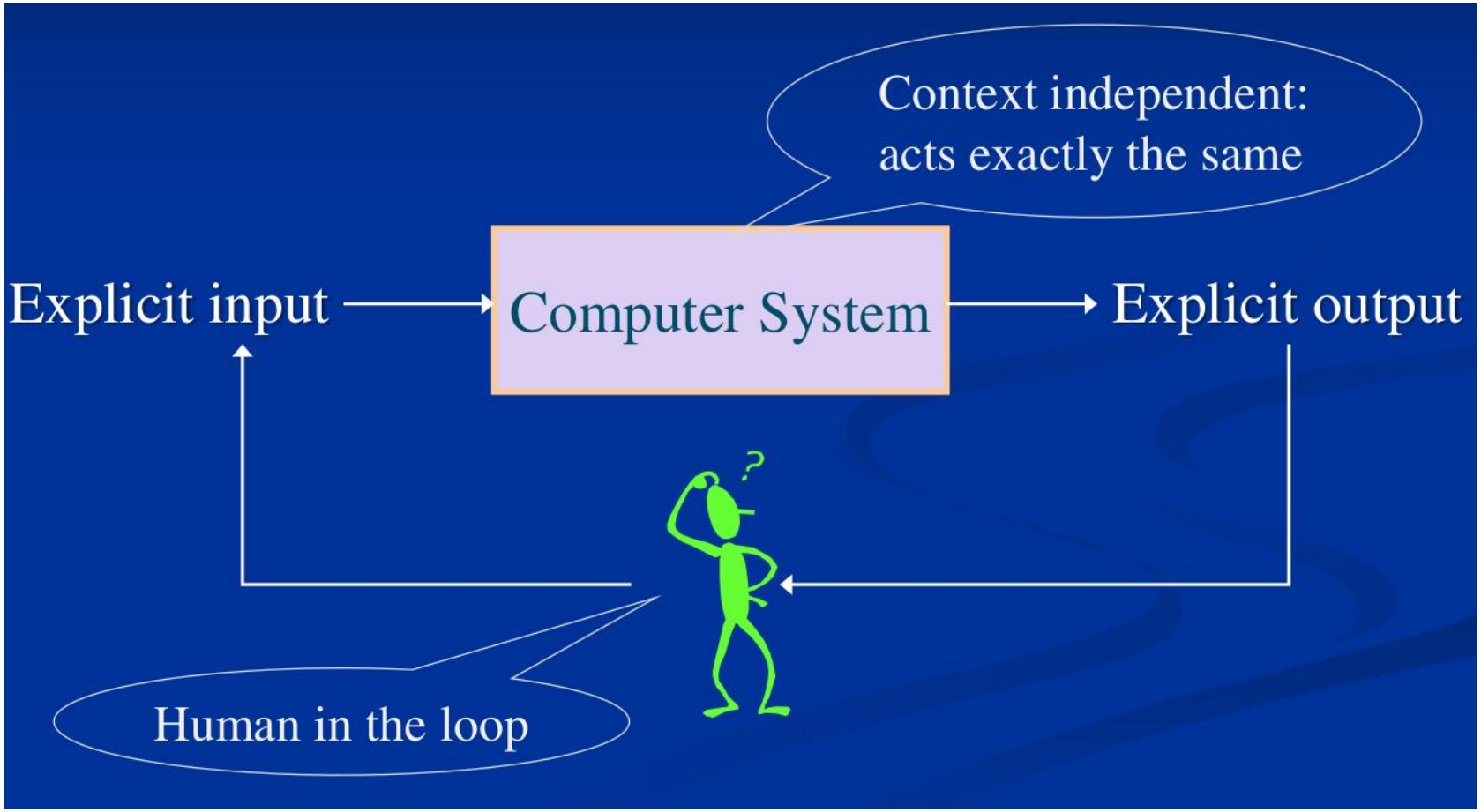
Physical Contexts: What, Where, and When

- Physical environment or phenomena
  - Temperature, light intensity, or chemical
- Location
  - Absolute or logical locations
- Time
  - Absolute or logical time

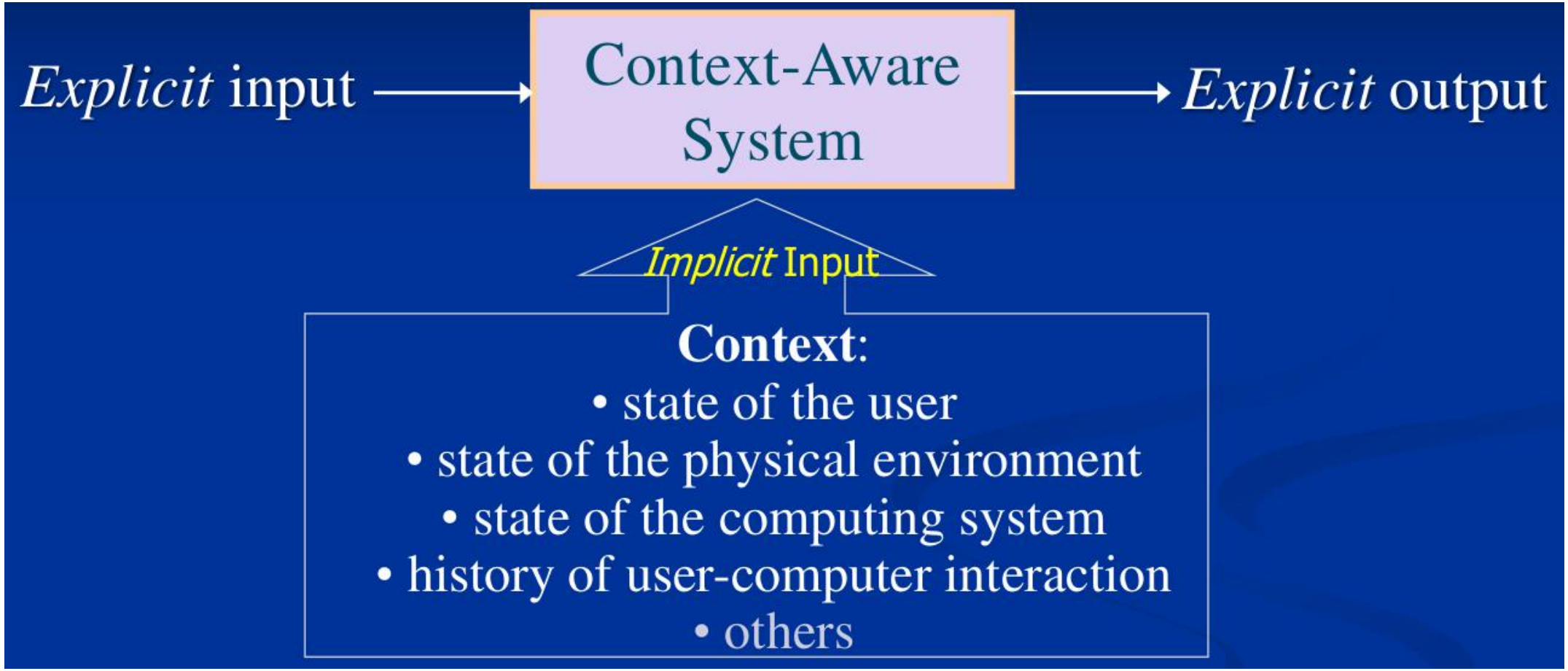
What is Context Aware Computing

- General awareness of the surrounding environment in which the user is operating, located, or situated
  - Context examples: user's preferences, likings, dislikes, location, etc
- Context-awareness is considered to be one of the fundamental properties of ubiquitous computing systems and is a key property of smart environments.

Traditional View of Computer Systems



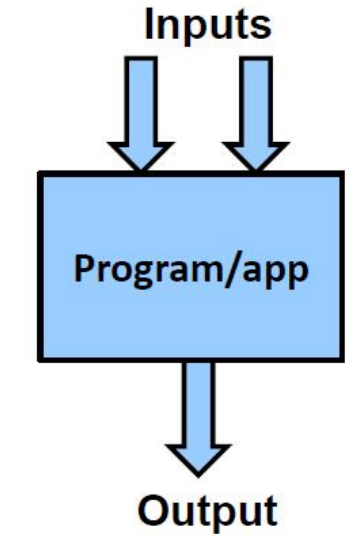
Context as Implicit Input/Output



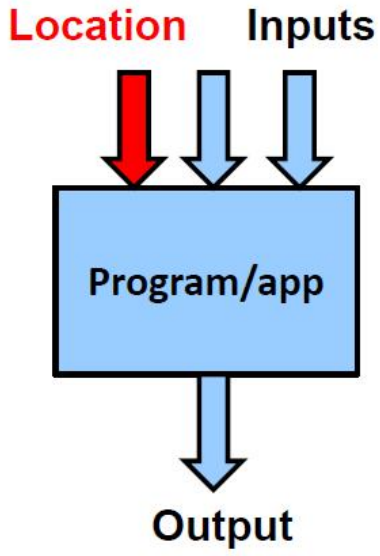
Context-aware computing examples

Example: Location-aware computing

- **Location-aware:** Location must be one of app/program's inputs
- Different user location = different output (e.g. maps)



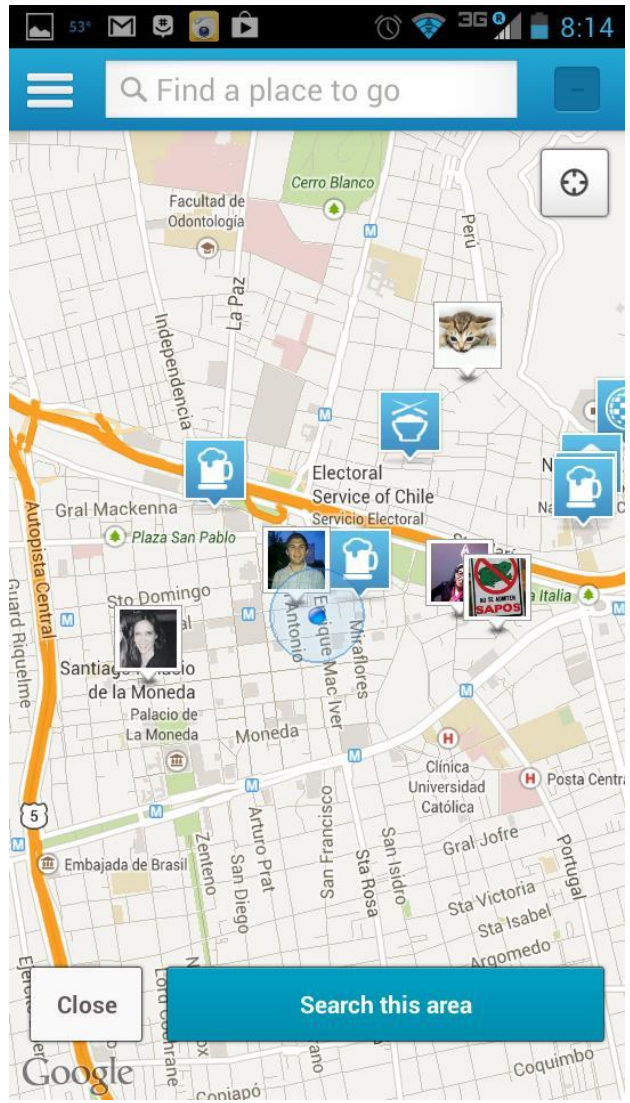
Non-mobile app



Mobile app

Location-Aware Computing

- Examples:
  - Map of user's "current location"
  - Print to "closest" printer
  - Apps that find user's friends "closeby"
  - Reviews of "closeby" restaurants



Reactive vs. Proactive LBS

- LBS can be either Reactive (“pull”) or Proactive (“push”)
- A Reactive LBS application is triggered by the user who queries the system in search of information based on the current location
- **Reactive LBS** examples
  - Finding restaurants or places of interest
  - Obtaining directions
  - Obtaining weather information
  - Sending emergency notifications to police, insurance companies, roadside assistance companies, etc.

Reactive vs. Proactive LBS

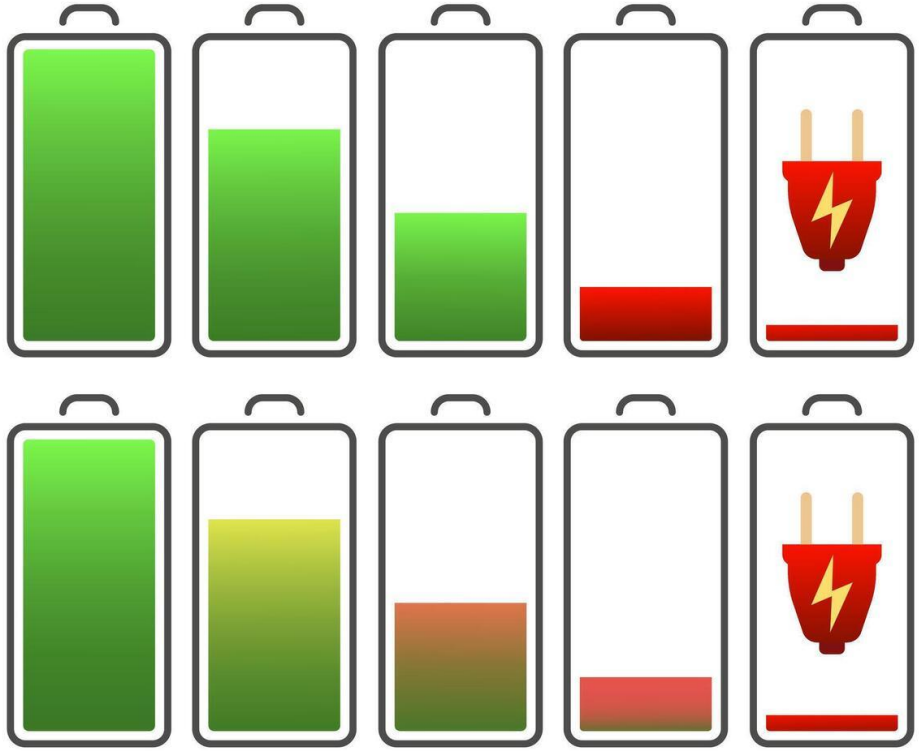
- In Proactive LBS applications, the system needs to continuously know where you are.
- Localization queries or actions are automatically triggered once a predefined set of conditions are met
- **Proactive LBS** examples
  - Geofencing, e.g., children outside predefined boundary
  - Fleet management
  - Real-time traffic congestion notifications
  - Real-time friend finding
  - Proximity-based actuation
  - Travel assistant device for riding public transportation, tourism, museum guided visits, etc

Questions: Proactive or Reactive?

- Location-based advertisement
- Payment based on proximity (EZ pass, toll watch)
- Play a radio station in local area
- App shows grocery list when near Walmart

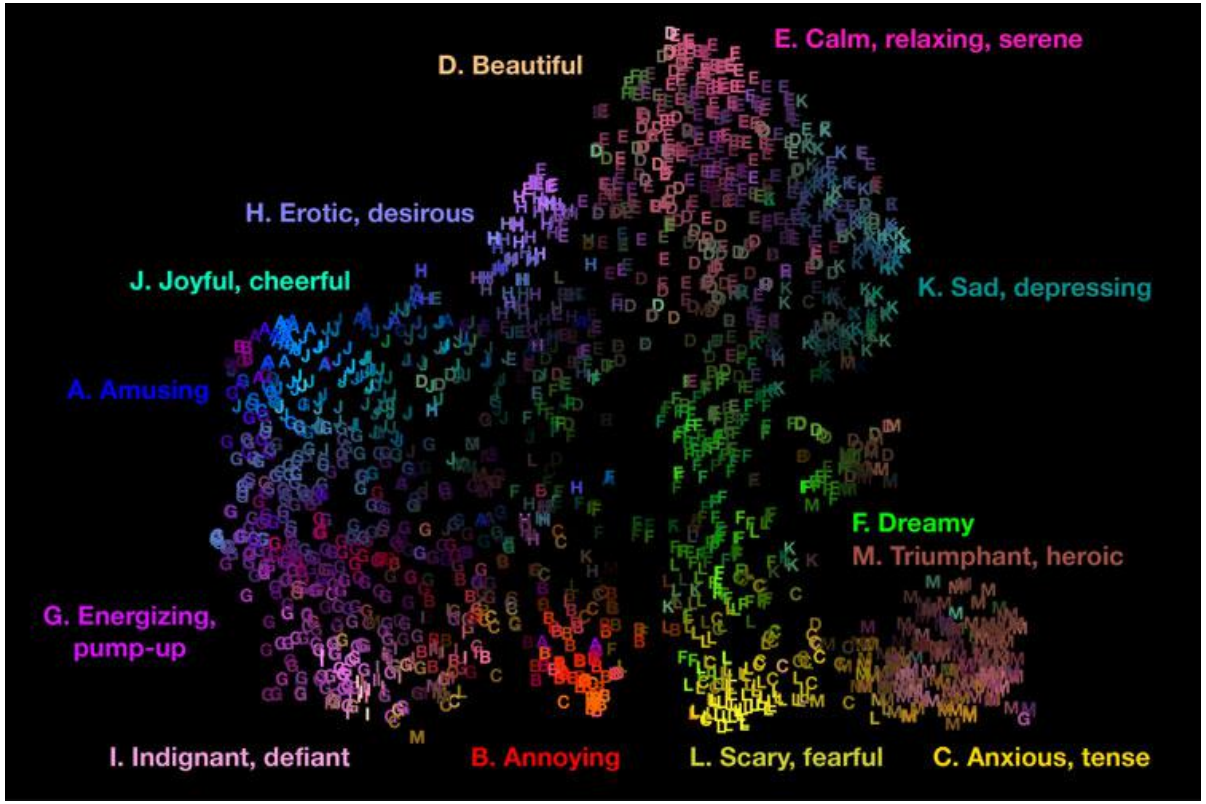
System Context

- System Awareness
  - how any context is created and adapted over an information and computing infrastructure
  - E.g., Wireless connectivity, Charging status



User context

- Consider a smart music player that adapts to user's mood



User Context

- Personal Context
  - **Preference.** E.g., a referee at a sports activity may prefer to blow the whistle for minor versus major sports offences.
  - **Identity.** E.g., owner vs guest
  - **Activity and Task.** E.g., running vs standing

- Social Context
  - how the actions of someone may affect others
  - E.g., who blows the whistle? The referee, policeman, or spectator?

Static versus Dynamic CA

- Static context
  - describes those aspects that are invariant
  - E.g., date of birth, User preference, home location, etc

- Dynamic context:
  - Information that changes change frequently
  - E.g., user activities, current locations, etc

Three Levels of Interactions for Context-Aware Infrastructure

1. Personalization: users recognizes the context and decide how the system should behave.
2. Passive context awareness: the system provides the user with context information; however the system does not automatically change behavior based on this.
   - Instead, the user decides on the course of action based on the updated context information.
3. Active context awareness: the system recognizes the context and takes required actions.
   - It offloads the work from the user by taking active decisions.

Discussion: Which CA Type?

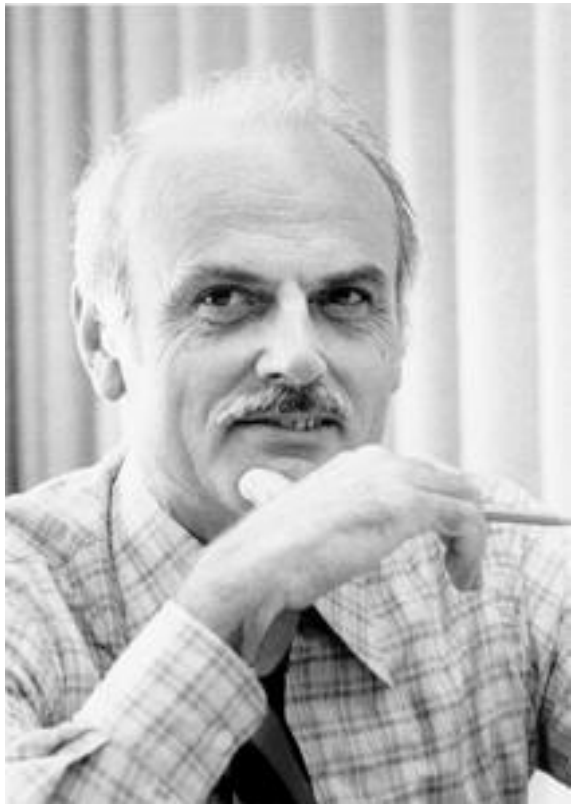
1. Personalization
2. Passive Context Awareness
3. Active Context Awareness

- A smart device uses location-based services to suggest dining locations, entertainment centers in the area, or even emergency services like hospitals and urgent care centers.
- A thermostat that changes temperature based on schedule
- The phone automatically disables notification when driving
- A smart watch could automatically adjust daylight savings or time zone based on the location context.

Mobile Database

Databases

- RDBMS
  - relational data base management system
- Relational databases introduced by
  - E. F. Codd
    - Turing Award Winner
- Relational Database
  - data stored in tables
  - relationships among data stored in tables
  - data can be accessed and viewed in different ways



Example Database

- Wine and region tables

Winery Table

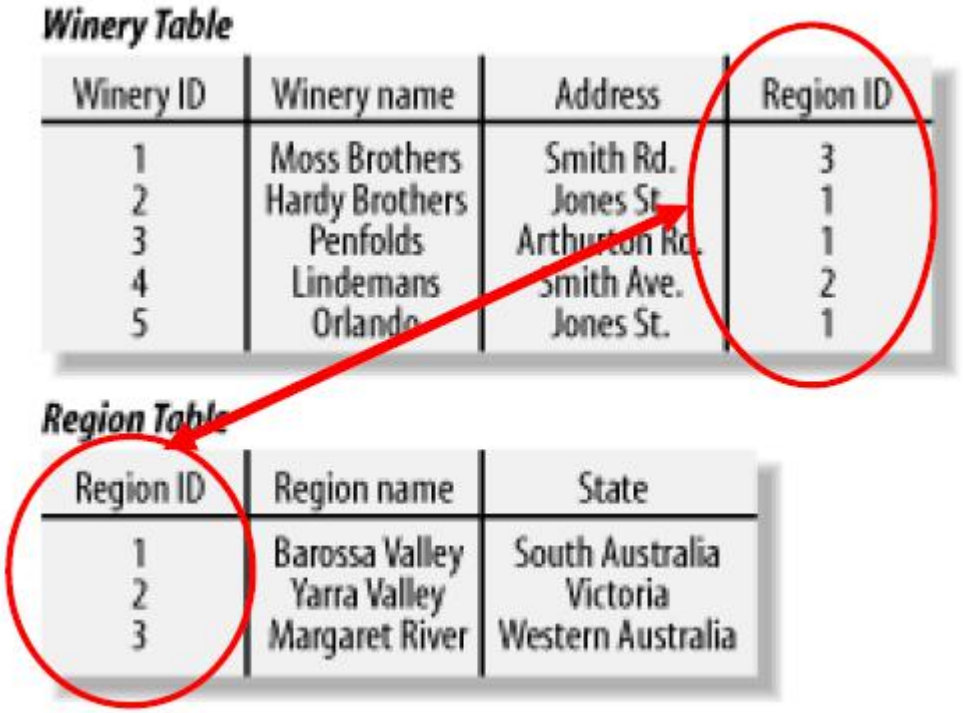
Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

Relational Data

- Data in different tables can be related



Winery Table

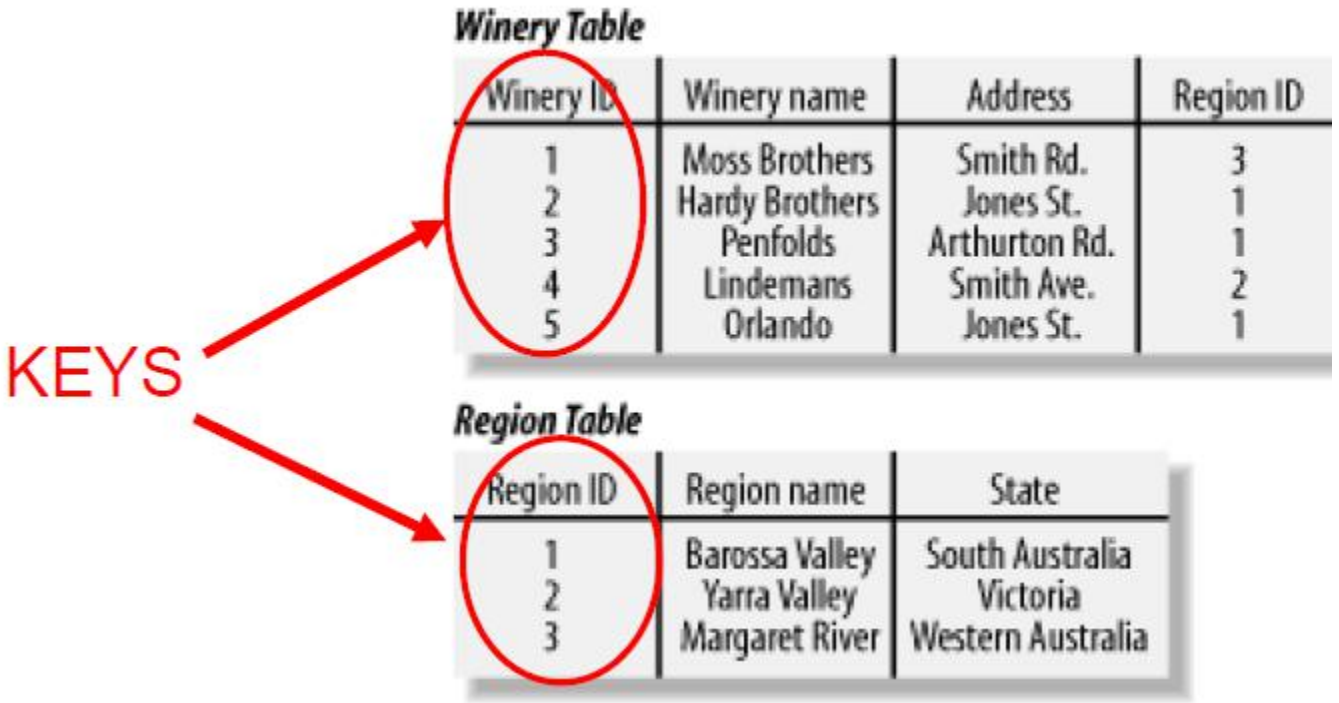
Winery ID	Winery name	Address	Region ID
1	Moss Brothers	Smith Rd.	3
2	Hardy Brothers	Jones St.	1
3	Penfolds	Arthurton Rd.	1
4	Lindemans	Smith Ave.	2
5	Orlando	Jones St.	1

Region Table

Region ID	Region name	State
1	Barossa Valley	South Australia
2	Yarra Valley	Victoria
3	Margaret River	Western Australia

Keys

- Each table has a key
- Column used to uniquely identify each row



SQL and Databases

- SQL is the language used to manipulate and
- manage information in a relational database
- management system (RDBMS)
- SQL Commands:
  - **CREATE TABLE** - creates new database table
  - **ALTER TABLE** - alters a database table
  - **DROP TABLE** - deletes a database table
  - **CREATE INDEX** - creates an index (search key)
  - **DROP INDEX** - deletes an index

SQL Commands

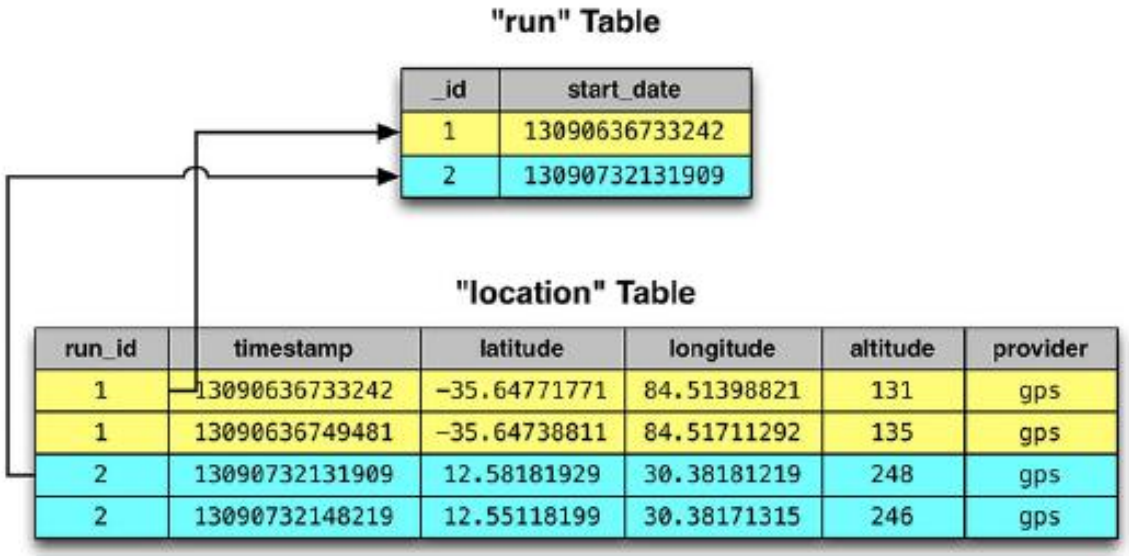
- **SELECT** - get data from a database table
- **UPDATE** - change data in a database table
- **DELETE** - remove data from a database table
- **INSERT INTO** - insert new data in a database table
- SQLite implements most, but not all of SQL
  - http://www.sqlite.org/

Why Local Databases?

- Data can get large
- Example: a running tracking app
  - What would such an app be like?
- User may track their runs forever
- **Solution:** Store runTracker runs and locations in SQLite database
  - **SQLite** is open source relational database
  - **SQLiteOpenHelper** encapsulates database creation, opening and updating
  - In **runTracker**, create subclass of **SQLiteOpenHelper** called **RunDatabaseHelper**

Use Local Databases in runTracker

- Create 1 table for each type of data
- Thus, we create 2 tables
  - **Run** table
  - **Location** table
- **Idea:** A run can have many locations visited



"run" Table

_id	start_date
1	13090636733242
2	13090732131909

"location" Table

run_id	timestamp	latitude	longitude	altitude	provider
1	13090636733242	-35.64771771	84.51398821	131	gps
1	13090636749481	-35.64738811	84.51711292	135	gps
2	13090732131909	12.58181929	30.38181219	248	gps
2	13090732148219	12.55118199	30.38171315	246	gps

Create RunDatabaseHelper

- **onCreate:** establish schema for newly created database
- **onUpgrade( ):** execute code to migrate to new version of schema
- Implement **insertRun(Run)** to write to database

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    public RunDatabaseHelper(Context context) {
        super(context, DB_NAME, null, VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Create the "run" table
        db.execSQL("create table run (" +
            "_id integer primary key autoincrement, start_date integer)");
        // Create the "location" table
        db.execSQL("create table location (" +
            " timestamp integer, latitude real, longitude real, altitude real," +
            " provider varchar(100), run_id integer references run(_id))");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Implement schema changes and data massage here when upgrading
    }

    public long insertRun(Run run) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_RUN_START_DATE, run.getStartDate().getTime());
        return getWritableDatabase().insert(TABLE_RUN, null, cv);
    }
}
```

Add ID to Run Class

- Add ID property to **Runs** class
- ID required to
  - Distinguish runs
  - Support querying runs

```
public class Run {
    private long mId;
    private Date mStartDate;

    public Run() {
        mId = -1;
        mStartDate = new Date();
    }

    public long getId() {
        return mId;
    }

    public void setId(long id) {
        mId = id;
    }

    public Date getStartDate() {
        return mStartDate;
    }
```

Use RunManager to use the database

Use this method when a new run Is STARTED (When Start button is Pressed)

Use this method when a new run Is RESTARTED

Use this method when Run Is STOPPED (When STOP button is Pressed)

```
private void broadcastLocation(Location location) {
    Intent broadcast = new Intent(ACTION_LOCATION);
    broadcast.putExtra(LocationManager.KEY_LOCATION_CHANGED, location);
    mAppContext.sendBroadcast(broadcast);
}

public Run startNewRun() {
    // Insert a run into the db
    Run run = insertRun();
    // Start tracking the run
    startTrackingRun(run);
    return run;
}

public void startTrackingRun(Run run) {
    // Keep the ID
    mCurrentRunId = run.getId();
    // Store it in shared preferences
    mPrefs.edit().putLong(PREF_CURRENT_RUN_ID, mCurrentRunId).commit();
    // Start location updates
    startLocationUpdates();
}

public void stopRun() {
    stopLocationUpdates();
    mCurrentRunId = -1;
    mPrefs.edit().remove(PREF_CURRENT_RUN_ID).commit();
}

private Run insertRun() {
    Run run = new Run();
    run.setId(mHelper.insertRun(run));
    return run;
}
}
```

- Example Usage

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
        Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_run, container, false);

    ...

    mStartButton = (Button)view.findViewById(R.id.run_startButton);
    mStartButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.startLocationUpdates();
            mRun = new Run();
            mRun = mRunManager.startNewRun();
            updateUI();
        }
    });

    mStopButton = (Button)view.findViewById(R.id.run_stopButton);
    mStopButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            mRunManager.stopLocationUpdates();
            mRunManager.stopRun();
            updateUI();
        }
    });

    updateUI();

    return view;
}
```

Inserting Locations into Database

- Similar to inserting runs, need to insert locations when **LocationManager** gives updates
- Add **insertLocation** method

```
public class RunDatabaseHelper extends SQLiteOpenHelper {
    private static final String DB_NAME = "runs.sqlite";
    private static final int VERSION = 1;

    private static final String TABLE_RUN = "run";
    private static final String COLUMN_RUN_START_DATE = "start_date";

    private static final String TABLE_LOCATION = "location";
    private static final String COLUMN_LOCATION_LATITUDE = "latitude";
    private static final String COLUMN_LOCATION_LONGITUDE = "longitude";
    private static final String COLUMN_LOCATION_ALTITUDE = "altitude";
    private static final String COLUMN_LOCATION_TIMESTAMP = "timestamp";
    private static final String COLUMN_LOCATION_PROVIDER = "provider";
    private static final String COLUMN_LOCATION_RUN_ID = "run_id";

    ...

    public long insertLocation(long runId, Location location) {
        ContentValues cv = new ContentValues();
        cv.put(COLUMN_LOCATION_LATITUDE, location.getLatitude());
        cv.put(COLUMN_LOCATION_LONGITUDE, location.getLongitude());
        cv.put(COLUMN_LOCATION_ALTITUDE, location.getAltitude());
        cv.put(COLUMN_LOCATION_TIMESTAMP, location.getTime());
        cv.put(COLUMN_LOCATION_PROVIDER, location.getProvider());
        cv.put(COLUMN_LOCATION_RUN_ID, runId);
        return getWritableDatabase().insert(TABLE_LOCATION, null, cv);
    }
```

Continuously Handle Location Updates

- System will continuously give updates
- Need to receive location intents whether app is visible or not
- Implement **dedicated Location Receiver** to insert location
- Inserts location into run whenever new location is received

```
public class TrackingLocationReceiver extends LocationReceiver {

    @Override
    protected void onLocationReceived(Context c, Location loc) {
        RunManager.get(c).insertLocation(loc);
    }

}
```

Alternatives to sqlite

- SQLite is low level ("Down in the weeds“)
- Various alternatives to work higher up the food chain
- Object Relational Mappers - ORM
- Higher level wrappers for dealing with SQL commands and sqlite databases
- Many ORMs exist